MASTER EN SCIENCES INFORMATIQUES

Planification de productions

Leroux, Marc-Antoine

Award date:
2006

Awarding institution:
Universite de Namur

Facultés Universitaires Notre-Dame de la Paix, Namur
Institut d'Informatique
Année académique 2005-2006

Planification de productions

Marc-Antoine Leroux

Mémoire présenté en vue de l'obtention du grade de Licencié en Informatique

Remerciements

En signe d'attachement aux Facultés universitaires Notre-Dame de La Paix, je dédie respectueusement ce travail à son Recteur, le Père Scheuer, s.j. En déposant ce mémoire, je tiens à remercier toutes celles et ceux qui m'ont aidé à le réaliser.

Mon Promoteur, le Professeur Leclercq a piloté ce projet et m'a maintes fois conseillé et orienté dans la rédaction de ce manuscrit. Ses suggestions et critiques toujours si pertinentes ont utilement influencé mon analyse. Qu'il trouve ici toute l'expression de ma gratitude.

Plusieurs membres du personnel du Groupe Solvay ont contribué de manière majeure à l'aboutissement du présent document. Monsieur Frédéric Tribel fût à la source même de ce mémoire puisqu'il m'a proposé d'emblée ce passionnant sujet. Le Docteur Wolfgang Lexa m'a fourni l'ensemble des éléments techniques qui constituent la trame de ce travail. Il m'aurait été impossible sans son aide de maîtriser les processus hautement complexes de la production dans un Groupe tel que Solvay. Monsieur Brieuc van den Hove m'a également initié aux principes de l'informatique industrielle et ne m'a jamais ménagé ni son temps ni son attention chaque fois que je lui ai demandé conseil. C'est un plaisir de leur témoigner ici ma plus profonde reconnaissance.

Enfin, l'ensemble des Professeurs de la Faculté d'Informatique m'ont permis d'acquérir des connaissances solides sans lesquelles la réalisation de ce travail aurait été impossible. Qu'ils veuillent trouver ici le témoignage de ma gratitude.

M-A Leroux
Bruxelles, décembre 2005

Résumé

Solvay est un groupe industriel basé à Bruxelles. Afin de réduire les coûts de la société, une optimisation efficace de la chaîne de production est rendue nécessaire. Pour atteindre ce but, nous proposons ici le développement d'un module paramétrisable de planification de production. Le principe d'optimisation de ce module repose sur l'algorithme du Simplex. Ce module s'intégrera dans le paysage informatique de Solvay. L'échange des données avec le système informatique de Solvay fût élaboré sous forme de fichiers XML. Plusieurs simulations nous ont permis de constater que les plans de production fournis par le modéliseur répondent aux contraintes imposées par la société. Nous suggérons donc que cette stratégie puisse s'avérer efficace dans la réduction des coûts totaux de production.

Solvay is an industrial company located in Brussels. In order to reduce the costs of the company, an efficient optimization of the supply chain must to be done. To achieve this target, we suggest the development of a supply planning module which fits to any continuous manufacturing. The principle of optimization is based on the Simplex Algorithm. This module will be integrated in the IT frame of Solvay. The data exchange with the Solvay IT system was implemented in XML files form. Many simulations showed us that the plans of production provided by our module fit the constraints imposed by the company. Then, we suggest this strategy which could be efficient for the reduction in the total costs of production.

Table des matières

Glossaire

- **APO** (*Advanced Planner and Optimiser*) : Nom d'un logiciel de planification de la chaîne logistique. Ce logiciel est développé par la société SAP.

- **DP** (*Demand Planning*) : Un module d'APO reprenant, entre autres, l'historique des ventes.

- **ERP** (*Enterprise Resources Planning*) : Logiciel de gestion intégré. Ensemble de modules (comptabilité, logistique, ressource humaine, ...) nécessaire à la gestion d'une entreprise.

- **GLPK** (*GNU Linear Programming Kit* : Bibiliothèque de fonctions permettant de résoudre un programme linéaire modélisé à l'aide de GNU MathProg. GLPK est un solveur.

- **GNU MathProg** : Bibiliothèque de fonctions permettant de modéliser un programme linéaire. GNU MathProg est un modeleur.

- **Horizon de planification** : Limite de temps sur laquelle notre plan de production fournit ses prévisions. En général, l'horizon de planification sera de 30 jours. L'unité minimum de l'horizon sera toujours le jour.

- **Minimum semi continu** : Une contrainte qui admet un minimum semi-continu doit, soit être nulle, soit dépasser un minimum donné.

- **Modeleur** : Logiciel qui permet de modéliser sous forme numérique un problème de programmation linéaire.

- **PP/DS** (*Production Planning/Detailed Scheduling*) : Module d'APO fournissant un planning de fabrication quotidien. Ce module ne fonctionnant pas bien, il sera remplacé au sein du système informatique de Solvay par le module développé dans ce travail.

- **PPM** (*Production Process Model*) : Structure de données décrivant le mode opératoire d'une fabrication.

- **R/3** : Version 3 du progiciel de gestion intégré de la société SAP. R/3 est un ERP.

- **Ressource** : Structure de données décrivant une unité de fabrication.

- **SAP** (*Système Applications and Product for data processing*) : Editeur de logiciel allemand. Souvent, sous la dénomination SAP, il faut entendre le logiciel intégré R/3.

- **SCM** (*Supply Chain Management*) : Gestion de la chaîne logistique.

- **SNP** (*Supply Network Planning*) : Un module d'APO calculant une planification à moyen terme.

- **Solveur** : Logiciel qui permet de résoudre un problème de programmation linéaire modélisé sous forme numérique. linéaire.

Chapitre 1

Introduction

Le Groupe Solvay est un groupe chimique et pharmaceutique qui se positionne en tant que leader mondial dans la plupart de ses activités.

Comme toute grande entreprise, Solvay consacre une part importante de son énergie à recevoir les commandes, gérer les stocks, planifier la production, s'approvisionner chez les fournisseurs, ... Toutes ces activités nécessitent, pour être efficaces, une optimisation généralisée connue sous le vocable de Supply Chain Management (SCM).

On entend sous le terme SCM (Supply Chain Management, ou en français GCL, gestion de la chaîne logistique) les outils et méthodes visant à améliorer et à automatiser l'approvisionnement en réduisant les stocks et les délais de livraison. On parle ainsi de travail en "flux tendu" pour qualifier la limitation au minimum des stocks dans toute la chaîne de production.
Les outils de la SCM s'appuient sur des informations de capacité de production présentes dans le système d'information de l'entreprise pour passer automatiquement des ordres de commandes. Ainsi, les outils de SCM sont très fortement corrélés au Progiciel de Gestion Intégré (ERP, Enterprise Resource Planning) de l'entreprise.
Idéalement, un outil de SCM permet de suivre le cheminement des pièces (on parle de traçabilité) entre les différents intervenants de la chaîne logistique.
La logique du flux tendu est au coeur de la SCM. Cependant, cette logique n'est ni naturelle ni simple d'application. Il s'agit d'une philosophie ou d'un état d'esprit qui cherche à éliminer le gaspillage de temps et des ressources. Le flux tendu réduit le niveau de stock et augmente la rotation de ce dernier en essayant de produire le bon produit en bonne quantité et au bon moment. Sa mise en oeuvre est délicate mais conduit souvent à de grandes améliorations (Guéret et ass., 2000).

Chez Solvay, l'ensemble de la SCM a été optimisé grâce à un outil informatique. Cependant, la planification de production pose de nombreux problèmes. C'est donc ce dernier point que nous souhaitions optimiser ici.

Pour ce faire, nous développerons un module paramétrisable permettant de réduire au maximum les coûts de production en fournissant un plan de production optimisé. En fonction d'un plan directeur de production, des besoins des clients internes, des besoins des clients externes, de l'évolution mensuelle des stocks et des importations éventuelles de produits finis, l'outil devra fournir un planning de fabrication journalier optimal. Ce module sera adaptable à tous types de production continue.

Les objectifs de cette optimisation sont d'améliorer les performances de la Supply Chain de Solvay en :

- Réduisant le temps administratif nécessaire à l'établissement d'une planification ;
- Augmentant la réactivité de la planification ;
- Diminuant les niveaux de stock et donc les coûts de stockage ;
- Diminuant les ventes non réalisées (qui n'ont pu être effectuées à cause d'un manque de quantité disponible) ;
- Respectant mieux les dates de livraison chez le client ;

Afin donc de poursuivre l'automatisation de la SCM dans l'entreprise, nous avons développé un module de prévision à court terme de la fabrication en continu. Ce module s'intègre dans le système informatique existant.

Dans ce travail, nous nous pencherons dans une première partie sur les processus communs à la fabrication chimique industrielle en modélisant les différentes étapes de cette production ainsi que les paramètres et contraintes associés aux différents éléments d'une chaîne de production. Ce sont évidemment ces contraintes et paramètres qui seront analysés par l'optimiseur afin de fournir un plan de production optimisé.

Ensuite, ce modèle de production a été traduit sous forme d'équations mathématiques solubles par l'algorithme du Simplex. Nous avons alors étudié plusieurs outils informatiques permettant de résoudre un système d'équation à l'aide de l'algorithme du Simplex. Notre choix s'est porté sur GLPK.

Afin de démontrer l'applicabilité de notre solution, nous avons fourni un plan de production optimum pour la fabrication du chlorure de vinyle chez Solvay. Afin de démontrer l'aspect généralisable de notre solution, nous l'avons ensuite appliqué à un autre type de production, celui de la soude.

La dernière partie de ce travail s'est intéressée à l'évaluation de l'outil en explicitant ses avantages et ses inconvénients.

Chapitre 2

Le groupe Solvay

2.1 Présentation du groupe

Les origines du groupe Solvay remontent à 1861 lorsque Ernest Solvay met au point un procédé de fabrication de la soude à partir d'ammoniaque. L'exploitation industrielle débute à Couillet deux ans plus tard. La croissance est fulgurante. Déjà avant la fin du siècle, Solvay produit la soude caustique et le chlore à partir de l'électrolyse du sel. Le groupe emploie 20.000 personnes en 1913.

Aujourd'hui, Solvay est un groupe chimique et pharmaceutique international qui se positionne en tant que leader mondial dans la plupart de ses activités.

Pour permettre au Groupe de s'adapter aux changements rapides du monde qui l'entoure et d'accélérer sa croissance, sa stratégie d'évolution s'articule selon deux axes :

- Renforcement du leadership dans toutes les activités, en développant constamment leur compétitivité et l'innovation.
- Développement plus rapide du Secteur Pharmaceutique et des Spécialités.

Solvay est actif dans 50 pays et 400 établissements. Plus de 30.000 salariés du Groupe, dont 2.500 sont engagés dans la recherche, s'appliquent à satisfaire 160.000 clients.

Le chiffre d'affaires au premier trimestre 2005 a atteint 2.092 millions d'EUR et est en hausse de 14% par rapport au premier trimestre 2004. Le chiffre d'affaires (en EUR) des trois secteurs progressent : plastiques (+18%), chimique (+15%) et pharmaceutique (+4%).

Actuellement, 85 % du chiffre d'affaires sont réalisés par des produits dans lesquels le Groupe figure parmi les tout premiers producteurs mondiaux.

Le Groupe est convaincu que la citoyenneté de l'entreprise est la clé d'une croissance durable. La mise en oeuvre à l'échelle mondiale d'une politique de Responsible Care®, destinée à assurer des progrès constants dans les domaines de la santé, de la sécurité et de l'environnement, constitue un des volets des efforts consentis par Solvay pour concilier le progrès socio-économique et la poursuite de ses activités à long terme.

Chez Solvay, l'ensemble de la SCM a été optimisé grâce à un outil informatique. Cependant, la planification de production pose de nombreux problèmes. C'est donc ce dernier point que nous souhaitons optimiser ici.

2.2 Le système informatique chez Solvay

Solvay s'est équipé d'un système d'information intégré qui couvre toutes les fonctions de la société (Planification, Production, Administration des ventes, Maintenance, Comptabilité,...) et ce dans le but de rencontrer plusieurs objectifs :

- permettre une meilleure visibilité du marché et réagir rapidement à ses fluctuations ;
- anticiper et répondre en temps réel aux attentes des clients ;
- faire franchir aux processus de gestion les limites de l'entreprise de façon à intégrer les clients, les fournisseurs et les partenaires.

Le système choisi par Solvay est basé sur la solution MySAP.com fournie par SAP AG [1].

Fondé en 1972, SAP est le premier fournisseur mondial de solutions business collaboratives pour toutes les entreprises.

Avec 12 millions d'utilisateurs, 84 000 installations, et 1 500 partenaires, SAP est le premier fournisseur mondial de logiciels inter-entreprises, et le troisième fournisseur mondial de logiciels.
SAP emploie aujourd'hui près de 30 000 personnes dans plus de 50 pays. SAP met au service de ses clients un réseau mondial de compétences, de processus et de produits, garantissant leur réussite.
Etablie à Walldorf en Allemagne, SAP est coté sur plusieurs marchés finan-

[1] "SAP" et "MySAP.com" sont des marques déposées par SAP Aktiengesellschaft, System, Applications and Products in Data Processing, Neurostrasse 16, 69190 Walldorf, Germany.

ciers, notamment aux bourses de Francfort et de New York, sous le symbole "SAP".

La solution MySAP.com est composée de 4 couches (Knolmayer et ass.,2001) :
- la couche transactionnelle pour la gestion de production, des achats, stocks et approvisionnements, des ventes et de la distribution ;
- la couche Supply Chain Execution pour la gestion de l'entreposage et du transport ;
- la couche d'APS (Advanced Planning System), pour l'élaboration des prévisions, la planification de production et du transport, l'ordonnancement, le calcul de disponibilité des produits, la modélisation et l'optimisation du réseau logistique, etc ;
- une couche de collaboration (échanges de prévisions de ventes, plans d'approvisionnements et de productions partagés...).

Le domaine fonctionnel que nous utiliserons dans le cadre de ce projet est la planification de la chaîne logistique. C'est le système APO (Advanced Planner and Optimizer) de SAP qui fournit les différents outils de planification pour nous aider à gérer une chaine logistique performante. Tous les jours, pour chaque produit, nous devons connaître la quantité disponible à la vente, la quantité à produire ainsi que la quantité à acheter.

APO gère le processus de planification depuis la collecte des données nécessaires à l'obtention d'une prévision des ventes jusqu'à la planification de production, et ce y compris l'ordonnancement et la distribution.

Au sein d'APO, trois différents modules sont distingués :

La *planification des ventes* engendre des décisions au niveau stratégique et est modélisée dans le module **DP** (Demand Planning). Le but de ce module est de fournir des prévisions de vente mensuelle sans contrainte avec un horizon d'un à deux ans. DP anticipe la demande et calcule les prévisions de vente sur base de l'historique des ventes.
La responsabilité des prévisions de vente incombe aux vendeurs, au département du marketing ainsi qu'aux responsables des "front offices".
Les vendeurs ont un contact direct avec les clients. Ce sont eux qui rendent visite aux clients pour les informer sur les produits de Solvay. Le "front office" est le département en charge de gérer les contacts indirects avec les clients. Il est responsable du traitement des commandes des clients appartenant à une région donnée.

La *planification à moyen terme*, correspond au niveau tactique et est modélisée pas **SNP** (Supply Network Planning). SNP optimise la production et la distribution interne dans tout le réseau de la chaîne logistique.

La responsabilité de la planification à moyen terme incombe au "Product Flow Manager". C'est lui qui va définir les niveaux de production et d'approvisionnement au niveau européen en fonction des prévisions de ventes. Il définit les allocations des productions vers les marchés/pays ("front office") concernés et coordonne les prévisions des productions locales en effectuant les arbitrages entre sites de production. Il y a un "Product Flow Manager" par unité organisationnelle en charge d'un marché.

La *planification à court terme* déterminant les décisions opérationnelles est modélisée dans **PP/DS**. PP/DS fournit un planning de fabrication quotidien et permet de faire l'ordonnancement des opérations. PP/DS est confiné au niveau local.
La responsabilité de la planification à court terme incombe au "Site Production Scheduler". C'est lui qui va établir la programmation de production de son site en fonction du planning mensuel qu'il a recu de son "Product Flow Manager". Il y a un "Site Production Scheduler" par site.

Le projet développé ici tentera d'améliorer la planification à court terme dans PP/DS en y intégrant un optimiseur. En effet, APO PP/DS ne tient compte ni des fabrications continues[2], ni des variations de productions, ni des minima de productions, ni des données fournies par SNP et fournit donc des résultats inexploitables. De plus, APO ne nous permet pas de travailler avec des processus incluant des recyclages (comme, par exemple, l'HCl lors de la production du VC ou la soude légère dans la fabrication de la soude (cfr. infra)), or ces processus sont très fréquents en chimie. Le projet ne s'intéressera qu'aux processus qui travaillent en production continue.

Actuellement, le système de planification utilisé par Solvay est purement manuel. Il repose essentiellement sur l'expérience des utilisateurs et ne fait l'objet d'aucune évaluation.

[2]APO proposera un plan de production avec des jours où il ne faut rien produire (production maximum le lundi, puis rien pour le reste de la semaine), ce qui est impossible à réaliser dans le cadre d'une production continue.

Chapitre 3

Modélisation des problèmes de production

3.1 Principes généraux d'une production

La production chimique industrielle fait intervenir des composés chimiques qui sont transformés afin d'obtenir d'autre composés chimiques. Chaque composé peut être dans un type de réaction le réactif et dans un autre type de réaction le produit, ces réactions pouvant se produire en chaîne.

Exemple 3.1.1 $NaCl + H2O \rightarrow NaOH + Cl2 \rightarrow NaClO + H2O$
Dans cette réaction, on comprend que le NaOH est à la fois produit de la réaction 1 et réactif de la réaction 2.

Dans la modélisation d'une production industrielle, il n'y a donc pas de sens à distinguer les matières premières et les produits finis. Les deux entités sont regroupées sous le vocable **Product**. A chaque produit est associée une série de contraintes qui lui sont propres et qui sont décrites ci-après.

C'est le mode opératoire, c'est-à-dire le processus de fabrication, repris sous l'abréviation **PPM**[1], qui permet de distinguer les matières premières des produits finis. En effet, pour chaque mode opératoire, on distingue les **PPMin** pour les matières premières et les **PPMout** pour les produits finis.

Pour chaque produit, il existe une demande pour chaque jour de l'horizon de planification, qui sera reprise dans notre schéma sous le vocable **Demand**. Celle-ci peut éventuellement être nulle ce qui est presque toujours le cas pour les matières premières [2].

[1] Production Process Model.

[2] Il arrive que Solvay fasse des opérations d'achats-vente, c'est-à-dire vende un produit acheté directement sans lui apporter aucune transformation. Dans ce cas nous aurons une demande en matières premières.

La production se fait sur un ensemble de machines reprises sous le vocables **Resource**. Chacune de ces ressources présente également une série de contraintes. Une même ressource peut être partagée par différents modes opératoires. Par contre, un mode opératoire fonctionnera toujours sur la même ressource.

Une ressource peut être en incapacité de production (**Incapacity**) pendant un temps donné et un jour donné de l'horizon par exemple pour des raisons d'entretien.

En fonction des réglages une ressource peut produire différentes quantités de produits [3]. Pour résoudre ce problème en une seule exécution de l'optimiseur, il suffit d'encoder les différents réglages comme différents PPMs.

Exemple 3.1.2 *Ainsi pour la production du chlorure de méthylène CLM2 (CH_2Cl_2) [4] et du chloroforme CLM3 ($CHCl_3$) [5] l'unité de fabrication peut donner, en fonction de son réglage, les productions suivantes :*

	Réglage 1	*Réglage 2*
Input Cl2 (tonnes)	*100*	*100*
Output CLM2 (tonnes)	*75*	*35*
Output CLM3 (tonnes)	*20*	*40*
Output ClH (tonnes)	*5*	*25*

Si l'on désire produire 30 CLM2 et 30 CLM3, il nous suffira d'entrer dans notre modèle un PPM représentant le réglage 1 et un autre PPM représentant le réglage 2. Notre modèle résoudra alors le système à deux inconnues suivant :

$$\begin{cases} 75x_1 + 35x_2 = 30 \\ 20x_1 + 40x_2 = 30 \end{cases} \quad (3.1)$$

et nous proposera de mettre en entrée 6,9 kg de CL2 avec le réglage 1 et 71 kg avec le réglage 2.

Afin que le module développé dans ce travail puisse fournir un plan de production, une série de paramètres et de contraintes devra lui être spécifiée. Ces paramètres seront détaillés dans la section **Entrées**. Ces paramètres sont :

- Les paramètres et contraintes sur les produits
- Les paramètres et contraintes liés aux modes opératoires

[3] Ceci n'est le cas ni pour la fabrication du VC ni pour la fabrication de la soude. Cette précision est faite afin de généraliser le modèle au maximum.

[4] Obtenu par chloration du chlorométhane : $CH_3Cl + Cl_2 \rightarrow CH_2Cl_2$.

[5] Obtenu par chloration du chlorure de méthylène : $CH_2Cl_2 + Cl_2 \rightarrow CHCl_3$.

- Les paramètres et contraintes liés aux ressources
- La demande en produit fini
- Les éventuelles incapacités de production

Grâce à l'ensemble de ces données, notre module d'optimisation créera un plan de production qui regroupe les informations suivantes, pour chaque jour :

- Niveau de stock pour chaque jour et chaque produit
- La quantité à acheter pour chaque produit
- Le niveau de production pour chaque produit
- Le niveau de demande satisfaite pour chaque produit
- Le niveau de demande non satisfaite pour chaque produit
- Le niveau d'utilisation de chaque ressource

Ces informations seront détaillées dans la section **Sorties**.

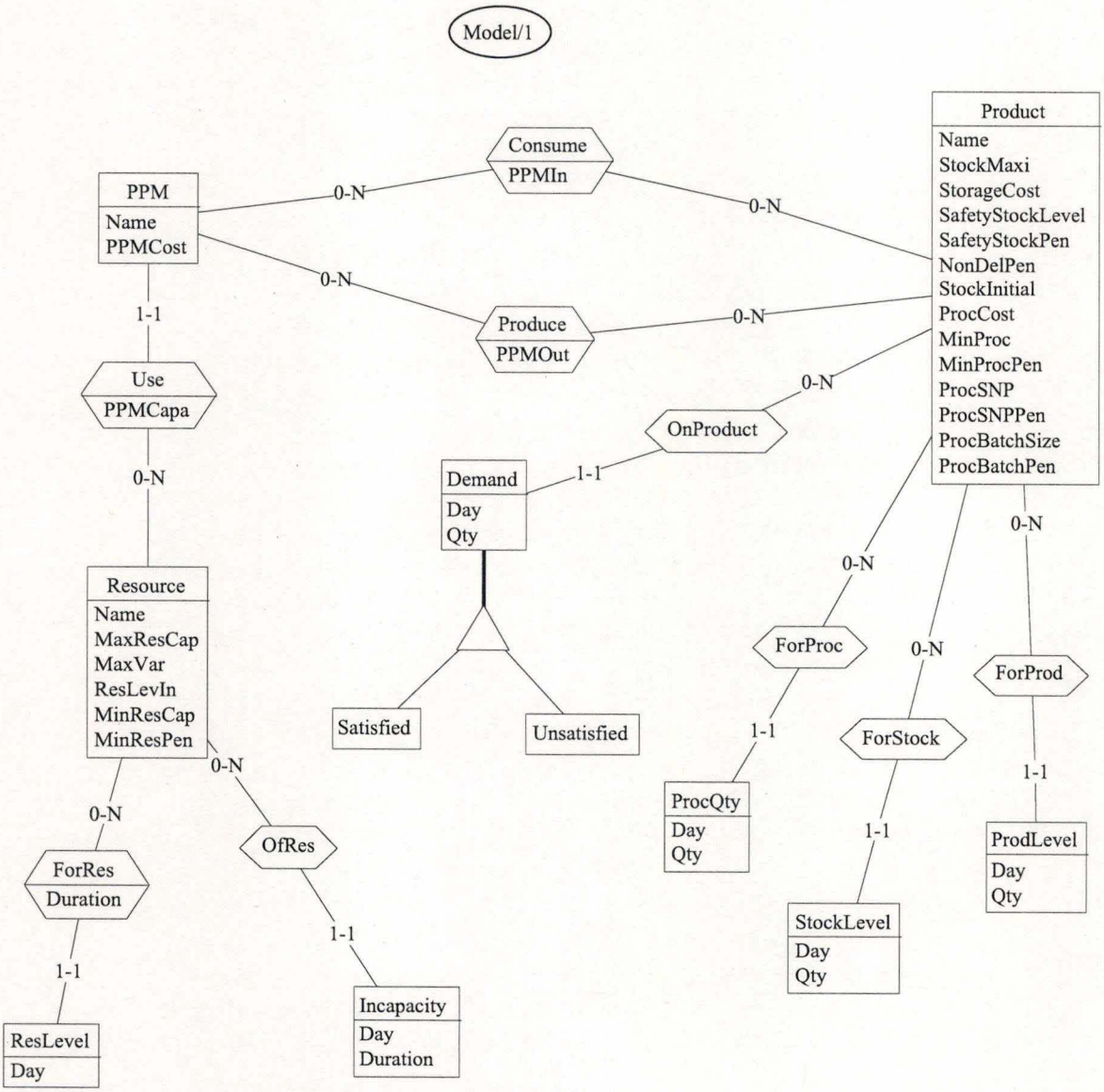

FIG. 3.1 - Modèle des données

3.2 Entrées

3.2.1 Les contraintes sur les produits(*Product)*

Pour chaque produit, les caractéristiques suivantes sont prises en compte :

L'identifiant du produit *(Name)*

Permet d'identifier de manière unique et non ambiguë chaque produit.

Le stock maximum *(StockMaxi)*

Le stock maximum est la quantité maximum de produit que l'on peut stocker. Cette quantité ne peut être physiquement dépassée. Il est donc inutile de déterminer un coût en cas de dépassement du stock maximum. Le stock maximum est déterminé sur base de plusieurs types de contraintes.

- Les contraintes de qualités (qui interviennent peu chez Solvay) et qui sont dues à la dégradation dues produit dans le temps.
- Le critère de dangerosité qui nous interdit ou nous limite les quantités de stockage. Ainsi nous ne pouvons stocker ni le chlore ni l'acide chlorhydrique.
- Les contraintes physiques pures par exemple le nombre de silos, les entrepôts à disposition, etc.

Il est à noter que le stock maximum peut varier dans le temps. Par exemple, lorsqu'un train est sur site, ses wagons peuvent servir de zone de stockage, augmentant ainsi temporairement la taille des stocks. D'autre part, pour certains produits, le stock peut être invariablement nul car ces produits sont dangereux et ne peuvent être stockés.

Le coût de stockage *(StorageCost)*

Vu que le but de ce module est d'optimiser les coûts de production, il est nécessaire de prendre en compte les frais inhérents aux stockages des différents produits. Pour valoriser le stock, il faut tenit compte de quatre groupes d'éléments (Samii 2001) :

- Les coûts de financement des investissements en stocks *("Capital stock")*. Garder du stock immobilise des capitaux qui pourraient être utilisés plus judicieusement. Le coût de financement des investissements en stock est le coût d'opportunité de l'argent multiplié par la valeur du stock. Le coût d'opportunité de l'argent n'étant autre que le coût interne des capitaux pour des projets à faible risque. En fonction du type d'entreprise, la valeur du stock, sera soit la valeur de remplacement,

soit la valeur du marché, soit la valeur d'aquisition.

- Les coûts associés à la tenue de stock (*"Inventory Service Stock"*). Ces coûts comprennent les frais d'assurance contre le vol et incendie majorés de la valeur de taxes éventuelles spécifiques à certains stockages, les frais d'entretien du dépôt, les coûts de chauffage, les coûts du personnel et les coûts de manipulation.

- Les coûts associés à l'espace de stockage (*"Storage space cost"*). Ce sont les côuts de location des entrepôts de stockage.

- Les coûts associés aux risques sur stock (*"Inventory risk costs"*). Sont compris ici, l'obsolescence, les dégats, les déperditions et le risque de transbordement et de transfert interne.

Chez Solvay, les coûts de stockage sont déterminés par l'expérience.

Le stock de sécurité *(SafetyStockLevel)*

Le stock de sécurité permet d'absorber l'imprévisible et, par conséquent, d'éviter la rupture de stock (Gaspart 1997). Pour fixer un stock de sécurité, il importe de définir les risques qu'il est destiné à couvrir dans le cas particulier de l'article qu'il concerne : risque lié à la demande du client par rapport au délai de fabrication, risque lié aux possibles défaillances d'approvisionnement, aux pannes... Dans le cas où il y a plusieurs risques, on ne retiendra que le ou les principaux. Lorsque les flux sont suffisament continus, sans lots trop importants nous pouvons utiliser la formule suivante (Eymery 1997) :

$$S_e = K\sqrt{E_1^2 + E_2^2 + ...} \tag{3.2}$$

avec :
- S_e : stock de sécurité ;
- $E_1, E_2, ...$: écarts types en quantité des principaux risques contre lesquels on souhaite se couvrir ;
- K : coefficient donné par la loi normale (de Gauss), en fonction du taux de service qu'on se donne comme objectif.

Chez Solvay, les stocks de sécurité sont déterminés par l'expérience.

Pénalité associée au dépassement du stock de sécurité *(SafetyStockPen)*

L'optimiseur permet de ne pas respecter la contrainte du stock de sécurité car il s'agit d'une contrainte économique et non physique. Cependant,

une pénalité artificielle est créée et permet à l'optimiseur de prendre les inconvénients liés à un stock trop faible et inférieur au stock de sécurité.

Pénalité en cas de non-livraison *(NonDelPen)*

A nouveau, il s'agit d'une contrainte économique et non physique. Un retard ou une absence de livraison n'entraîne pas de coûts directs et immédiats mais s'accompagne d'un manque à gagner lié à la perte du bénéfice et également d'un déficit en terme d'image auprès du client. Il est donc indispensable que l'optimiseur prenne en compte ce facteur. A nouveau, la pénalité qui lui est assignée est artificielle. Les pénalités sont calculées pour permettre de réaliser l'optimisation en fonction des stratégies du business. Par exemple, si un produit A nécessite pour sa production un produit B mais que la vente du produit A est plus rentable, l'optimiseur peut favoriser la vente du produit A quitte à ne pas satisfaire la demande en produit B. Pour permettre cette optimisation, il faut que la pénalité en cas de non-livraison soit plus faible sur le produit B que sur le produit A.

Stock initial *(StockInitial)*

Il s'agit du niveau de stock du produit au jour 0 de l'horizon d'optimisation. Il est intégré aux contraintes sur les produits car il doit être spécifié par l'utilisateur à l'optimiseur. Cette donnée sera alors utilisée par l'optimiseur pour créer le plan de production.

Coût d'achat *(ProcCost)*

Le coût d'achat représente le montant engendré par l'approvisionnement en matière première. Certaines de ces matières premières doivent être obligatoirement achetées parce que ne se retrouvant à aucun niveau de la chaine de production. Par contre, d'autres produits peuvent être soit achetés soit produits. Le choix se fera en fonction des capacités de production et des coûts engendrés par une éventuelle non-livraison. Par exemple, un produit B est nécessaire pour fabriquer un produit A. La chaîne de production a atteint sa limite de capacité et ne peut plus fabriquer le produit B. Celui-ci peut alors être acheté sauf si le coût d'achat dépasse le coût de non-livraison.

Quantité minimum à acheter *(MinProc)*

Il s'agit de la quantité minimum à acheter. Par exemple, on fait livrer uniquement un produit par camion rempli entièrement. Il s'agit ici d'un minimum semi-continu, c'est-à-dire que, soit nous pouvons ne rien acheter, soit nous pouvons acheter au moins un minimum donné.

Pénalité en cas de non respect du minimum à acheter *(MinProcPen)*

L'optimiseur permet de descendre en-dessous de la quantité minimum à acheter mais ce franchissement est pénalisé. En effet, il n'est pas rentable d'affréter un camion qui pourrait transporter 20 tonnes pour lui faire acheminer seulement une tonne.

Achats obligatoires *(ProcSNP)*

Grâce à ce paramètre, l'optimisation prend en compte les contraintes du Groupe. En effet, le Groupe peut imposer l'achat de produit à un centre de production afin de limiter les pertes dans une autre usine. Ainsi, même si la production d'un produit intermédiaire peut être réalisée dans un centre de production, la direction peut imposer à ce centre d'acheter le produit à une autre usine. Ceci aura un coût plus élevé pour le centre de production mais s'inscrira dans une logique de rentabilité pour l'ensemble du Groupe.

Pénalité en cas de non respect des ordres d'achats obligatoires *(ProcSNPPen)*

Le non-respect d'un ordre d'achat obligatoire par un centre de production reste possible parce que négociable mais entraînera une pénalité.

Taille des lots de produit à acheter *(ProcBatchSize)*

Ce paramètre prend en compte la taille minimum d'un lot de produit.

Pénalité en cas de non respect de la taille des lots à acheter *(ProcBatchPen)*

La taille minimum des lots peut également être négligée par l'utilisateur mais entraine alors une pénalité. En effet, si un produit A s'achemine par camion de 20 tonnes, il logique que l'achat de 21 tonnes de produit entraîne une perte quant aux coûts de transport. Cette perte est donc prise en compte par l'optimiseur.

3.2.2 Les contraintes sur les modes opératoires *(PPM)*

Le mode opératoire représente la méthode de fabrication d'un ou plusieurs produits.

L'identifiant du mode opératoire *(Name)*

Permet d'identifier de manière unique et non ambiguë le mode opératoire.

Le coût d'utilisation du mode opératoire *(PPMCost)*

Il s'agit du coût engendré par l'ensemble de la fabrication des produits de ce mode opératoire.

3.2.3 Les contraintes sur la demande *(Demand)*

La demande représente la quantité à fournir pour un produit donné à un jour donné.

3.2.4 Demande de produit *(OnProduct)*

Il s'agit du lien entre la quantité demandée et le produit demandé.

3.2.5 Consomme *(Consume)*

Il s'agit de la quantité pour chacune des matières premières nécessaires au mode opératoire. Cette entité fait le lien entre un PPM et un produit(matière première).

3.2.6 Fabrique *(Produce)*

Il s'agit de la quantité de chacun des produits finis fabriqués par le mode opératoire. Cette entité fait le lien entre un PPM et un produit fabriqué.

3.2.7 Utilise *(Use)*

Ce lien représente le rendement de chaque ressource pour un mode opératoire donné. Il est exprimé en unité de temps nécessaire à la production d'une certaine masse de produit fini principal du mode opératoire. Cette masse sera la quantité fabriquée reprise dans le lien "Produce" du produit principal. Si coproduit il y a, le produit principal est celui dont la masse fabriquée est la plus grande.

3.2.8 Les contraintes sur les ressources *(Resources)*

L'identifiant de la ressource *(Name)*

Permet d'identifier de manière unique et non ambiguë la ressource.

Le maximum de production d'une ressource *(MaxResCap)*

Cette contrainte représente le rendement maximum de la ressource. Hormis les incapacités, toute ressource fonctionne 24h/24 en production continue. Cependant, pour des raisons de coût, on peut faire fonctionner une ressource à un rendement moindre. Dans notre modélisation, nous considérons alors que le rendement est toujours de 100% mais sur une durée inférieure à

24h. Ainsi, nous considérons identique une ressource fonctionnant 24 heures à 50% de son rendement et une ressource fonctionnant 12h à 100% de son rendement. La quantité massique fournie par la ressource dépend en plus du mode opératoire et donc ne peut être spécifiée dans un modèle général au niveau de la ressource. C'est pourquoi nous avons mis cette quantité massique au niveau du lien entre la ressource et son PPM ("Use").

Le minimum de production d'une ressource *(MinResCap)*

Une ressource n'est rentable que si elle produit un minimum donné. En-dessous de ce minimum, la rentabilité de la ressource est affaiblie. A nouveau, ce minimum est exprimé en unité de temps qui est égal à un certain pourcentage de 24 heures. Ce minimum est également un minimum semi-continu, soit la ressource ne fonctionne pas, soit elle fonctionne au-dessus d'un certain minimum.

Pénalité en cas de franchissement du minimum de production d'une ressource *(MinResPen)*

L'optimiseur permet une production inférieure au minimum de production d'une ressource. Cependant, cette situation est pénalisée afin que l'optimiseur tienne compte de cette contrainte.

Variation maximum des niveaux de production *(MaxVar)*

La variation des niveaux de production d'un jour à l'autre doit être faible afin de rentabiliser le temps de travail et l'équipement de production. De plus, certains équipements ne supportent pas des variations des niveaux de production trop importantes sous peine de détériorations. Il est exprimé en pourcentage de variations autorisé.

Niveau initial d'utilisation de la ressource *(ResLevIn)*

Il s'agit du niveau initial d'utilisation de la ressource au jour 0 de l'horizon de production. Cette donnée est intégrée aux contraintes concernant la ressource car elle doit être spécifiée par l'utilisateur au moment du lancement de l'optimisation.

3.2.9 L'incapacité de production *(Incapacity)*

Il s'agit du temps durant lequel la ressource est inutilisable pour des raisons par exemple techniques. Il est exprimé en heure.

3.2.10 De la ressource *(Of)*

Il s'agit du ou des jours durant le ou lesquels une incapacité concerne une ressource.

3.3 Sorties

3.3.1 Niveau de stock *(StockLevel)*

Il s'agit du niveau de stock pour un produit donné à un jour donné. Avec comme cas particulier le stock au jour zéro de l'horizon de planification, qui correspond au stock initial.

3.3.2 Demande satisfaite *(Satisfied)*

Il s'agit de la quantité vendue au client.

3.3.3 Demande non satisfaite *(Unsatisfied)*

Il s'agit de la quantité demandée par le client mais qui n'a pu être fournie.

3.3.4 Produits à acheter *(ProcQty)*

Il s'agit de la quantité de produits à acheter à un jour donné de l'horizon pour permettre la production. Il s'agira en général uniquement de matières premières, mais il peut arriver que si une usine n'a pas une capacité de production suffisante, il faille également acheter des produits finis.

3.3.5 Quantité produite *(ProdLevel)*

Il s'agit de la quantité produite d'un produit donné à un jour donné en unité massique.

3.3.6 Niveau d'utilisation d'une ressource *(ResLevel)*

Il s'agit du niveau d'utilisation à appliquer sur chaque ressource pour chaque jour de l'horizon de production. Cette donnée est exprimée en heure.

Chapitre 4

Modélisation mathématique

4.1 Introduction

Le but est de fournir des équations mathématiques qui permettent de calculer un plan de production au coût minimum tout en respectant au mieux les besoins exprimés.
Le programme devra minimiser au maximum :

- les coûts de production ;
- les coûts de stockage ;
- les coûts d'approvisionnement ;
- les pénalités en cas de descente en-dessous du stock minimum ;
- les pénalités en cas d'achat obligatoire non respecté ;
- les pénalités en cas d'achat inférieur à l'achat minimum ;
- les pénalités en cas de demande non satisfaite ;
- les pénalités en cas de franchissement du minimum de production ;
- les pénalités en cas de non respect de la taille des lots à acheter ;

4.2 La programmation linéaire

La programmation mathématique est le nom donné aux problèmes d'optimisation liée ou optimisation sous contraintes. Il s'agit de rechercher l'optimum d'une fonction de variables, étant donné que celles-ci doivent vérifier un certain nombre d'équations et/ou d'inéquations appelées contraintes. Le problème le plus simple de la programmation mathématique est celui de programmation linéaire (P.L) : il s'agit de la situation où à la fois la fonction à optimiser et les contraintes à respecter sont linéaires, c'est-à-dire du premier degré en les variables (Teghem, 1996). La programmation linéaire a été développée en 1947 par George Dantzig (Dantzig, 1948) lorsqu'il découvrit la méthode du Simplex.
Un programme linéaire a la forme générique suivante. Il comporte n variables

non négatives, m contraintes d'égalité ou d'inégalité et la fonction objectif z à optimiser. Le coefficient de coût ou de profit de la variable x_j est noté c_j, celui de la variable x_j dans la contrainte i est noté a_{ij}. La contrainte i a un second membre constant b_i.

$$Max \text{ ou } Min \text{ Z} = \sum_{j=1}^{n} c_j x_j$$

$$\forall i = 1..m : \sum_{j=1}^{n} a_{ij} x_j \leq , = \text{ ou } \geq b_i$$

$$\forall j = 1..n : x_j \geq 0$$

Des valeurs de variables qui vérifient toutes les contraintes forment une *solution réalisable.* Une solution réalisable est *optimum* si aucune solution n'a un profit supérieur.

Le lecteur interessé trouvera une description précise de la programmation linéaire dans *Dantzig, 1997* et *Dantzig 2003* ou dans *Wolsey 1998.* Pour une introduction à la recherche opérationnelle, nous lui conseillons *Faure et ass., 2000.*

4.3 Les données

Les ensembles

- *Jours* : Chaque jour dans l'horizon de planification est représenté par une variable j allant de 1 à m. La planification de production se fera sur un horizon de 30 jours (m=30).
- *Produits* : Chaque produit dont la planification est optimisée est représenté par une variable a allant de 1 à n. Dans le cadre de la fabrication du VC, il y a 5 produits (n=5), dans le cadre de la fabrication de soude, il y en a 6 (n=6).
- *Ressources* : Chaque ressource utilisée lors de productions est représentée par une variable r allant de 1 à p. Dans le cadre de la fabrication du VC, il y en a 3 (p=3) et pour la soude 6 (p=6).
- *PPMs* : Chaque PPM (mode opératoire) utilisé lors de productions est représenté par une variable b allant de 1 à o. Dans le cadre de la fabrication du VC il y a 3 PPMs (o=3) et dans celui de la soude 6 (o=6) [1].

[1] Dans le cadre des productions présentées dans ce travail o=p, cependant la distinction est importante pour permettre de traiter les cas où plusieurs PPMS différents utilisent une même ressource.

Les produits

- Le paramètre *StockMaxi(n)* est la quantité maximum stockable de l'article a.
- Le paramètre *StorageCost(n)* est le coût par unité de stock et par jour de l'article a. Il représente la valeur de l'argent immobilisé.
- Le paramètre *SafetyStockLevel(n)* est la quantité minimum de l'article a que l'on doit conserver en stock.
- Le paramètre *SafetyStockPen(n)* représente le coût économique en cas de franchissement en-dessous du stock minimum.
- Le paramètre *NonDelPen(n)* est le coût en cas de demande non satifsaite de l'article a, c'est à dire le coût associé en cas de non livraison.
- Le paramètre *ProcCost(n)* est le coût d'achat de l'article a.
- Le paramètre *StockInitial(n)* est le stock en début de période. Il sera initialisé à la valeur du Safety Stock, c'est-à-dire la situation idéale à laquelle on souhaite aboutir.
- Le paramètre *MinProc(n)* est la quantité minimum de l'article a à acheter si du produit a est acheté.
- Le paramètre *MinProcPen(n)* est la pénalité associée au non respect du minimum de l'article a à acheter si du produit a est acheté.
- Le paramètre *ProcSNP(n)* est la quantité d'achats obligatoires plannifiée par SNP.
- Le paramètre *ProcSNPPen(n)* est la pénalité par unité d'achats obligatoires non respectés.
- Le paramètre *ProcBatchSize(n)* est la taille d'un lot de produits à acheter.
- Le paramètre *ProcBatchPen(n)* est la pénalité associée au non respect des achats de lots de taille fixe de l'article a.
- Le paramètre M représente une valeur suffisamment grande nous permettant de résoudre le problème du minimum semi-continu[2] pour les quantités à acheter.

Les ressources

Une ressource est une unité de production.

- Le paramètre *MaxResCap(p)* est la quantité maximum de production de la ressource r. Ce maximum est exprimé en unité de temps.
- Le paramètre *MaxVar(p)* est la variation maximale qu'une ressource peut supporter d'un jour à l'autre. Cette variation est exprimée en pourcent.

[2]Nous appellons minimum semi-continu le cas où il ne faut, soit rien acheter, soit un minimum donné.

- Le paramètre *ResLevIn(p)* est le niveau de production enregistré le dernier jour avant le début de l'horizon de planification.
- Le paramètre *MinResCap(p)* est la production minimum qu'une ressource r doit fournir. Ce minimum est exprimé en unité de temps.
- Le paramètre *MinResPen(p)* est la pénalitée associée au non respect de la production minimum d'une ressource r.
- Le paramètre G représente une valeur suffisamment grande nous permettant de résoudre le problème du minimum semi-continu de capacité sur les ressources.
- Le paramètre *ResInc(p,m)* représente les périodes d'arrêts des ressources. Il est exprimé en temps d'incapacité de fonctionnement d'une ressource r au jour j.

Les PPMs

Les Production Process Model (PPMs) représentent le mode opératoire d'une fabrication. Un PPM se fait sur une ressource donnée et implique n produits (en entrée ou en sortie) possédant chacun un ratio.

- Le paramètre *PPMCost(o)* est le coût d'utilisation d'un ppm p, c'est-à-dire, les coûts de production de l'ensemble des articles présents dans le PPM p.
- Le paramètre *PPMCapa(o,p)* représente le temps qu'il faut pour produire une unité de produit principal sur la ressource r.
- Le paramètre *PPMIn(o,n)* représente la quantité d'un article a consommée par le ppm b.
- Le paramètre *PPMOut(o,n)* représente la quantité d'un article a produite par le ppm b.

La demande

- Le paramètre *Demand(n,m)* représente la demande totale d'un produit au jour j.

4.4 Les variables

- La variable *stocklevel(a,j)* donne la quantité stockée de l'article a au jour j. Le stock initial est pour chaque produit son safety stock.
- La variable *satdem(a,j)* représente la demande du produit a qui a été satisfaite au jour j.
- La variable *unsatdem(a,j)* représente la demande du produit a qui n'a pas été satisfaite au jour j.
- La variable *depdem(a,j)* représente la demande du produit a induite par une production d'un autre produit.

- La variable *proclevel(a,j)* représente la quantité du produit a que l'on achète le jour j.
- La variable *ppmlevel(b,j)* représente le taux d'utilisation d'une ppm b au jour j. Une unité de ppmlevel produit les quantités données dans le tableau des PPMs.
- La variable *prodlevel(a,j)* représente la quantité produite de l'article a au jour j.
- La variable *reslevel(r,j)* représente le niveau de production d'une ressource r (en unité de temps) au jour j.
- La variable $\alpha(a,j)$ représente le stock de sécurité virtuel positif du produit a au jour j.
- La variable $\beta(a,j)$ représente le stock de sécurité virtuel négatif du produit a au jour j.
- La variable $\gamma(a)$ représente la différence positive de quantité achetée par rapport aux prévisions de SNP du produit a sur l'ensemble de la période.
- La variable $\delta(a)$ représente la différence négative de quantité achetée par rapport aux prévisions de SNP du produit a sur l'ensemble de la période.
- La variable *minproclevel(a,j)* représente la quantité manquante du produit a réellement achetée au jour j par rapport au minimum du produit a à acheter.
- La variable $x(a,j)$ représente une variable binaire permettant de résoudre le problème du minimum semi-continu pour les quantités à acheter.
- La variable *numberofbatches(a,j)* représente le nombre de lots du produit a à acheter au jour j. Cette variable est une variable en nombres entiers.
- La variable *minbatchlevel(a,j)* représente la quantité manquante du produit a à acheter au jour j pour arriver à un lot complet.
- La variable *minreslevel(r,j)* représente la capacité manquante sur la ressource r au jour j pour atteindre la capacité minimum de la ressource r.
- La variable $y(r,j)$ représente une variable binaire permettant de résoudre le problème du minimum semi-continu sur les productions des ressources.

Précisons que toutes nos variables sont positives.

4.5 La fonction "objectif"

$$Z = Min(Z_1 + Z_2 + Z_3 + Z_4 + Z_5 + Z_6 + Z_7 + Z_8 + Z_9) \qquad (4.1)$$

La première partie de la fonction "objectif" représente les coûts de production.

$$Z_1 = \sum_{b=1}^{o} \sum_{j=1}^{m} ppmlevel_{b,j} \times PPMCost_b \tag{4.2}$$

La seconde partie de l'équation consiste à minimiser les coûts de stockage.

$$Z_2 = \sum_{a=1}^{n} \sum_{j=1}^{m} stocklevel_{a,j} \times Storagecost_a \tag{4.3}$$

La troisème partie représente les coûts en cas de franchissement du stock minimum.

$$Z_3 = \sum_{a=1}^{n} \sum_{j=1}^{m} \beta_{a,j} \times SafetyStockPen_a \tag{4.4}$$

La quatrième partie représente les coûts en cas de demande non satisfaite.

$$Z_4 = \sum_{a=1}^{n} \sum_{j=1}^{m} unsatdem_{a,j} \times NonDelpen_a \tag{4.5}$$

La cinquième partie représente les coûts d'achats.

$$Z_5 = \sum_{a=1}^{n} \sum_{j=1}^{m} proclevel_{a,j} \times ProcCost_a \tag{4.6}$$

La sixième partie représente les coûts en cas d'achats planifiés par SNP non respectés.

$$Z_6 = \sum_{a=1}^{n} \delta_a \times ProcSNPPen_a \tag{4.7}$$

La septième partie représente les coûts en cas de non respect sur la quantité minimum à acheter.

$$Z_7 = \sum_{a=1}^{n} \sum_{j=1}^{m} minproclevel_{a,j} \times MinProcPen_a \tag{4.8}$$

La huitième partie représente les coûts en cas de non respect de la taille des lots à acheter.

$$Z_8 = \sum_{a=1}^{n} \sum_{j=1}^{m} minbatchlevel_{a,j} \times ProcBatchPen_a \tag{4.9}$$

La neuvième et dernière partie calcule les pénalités si une ressource fonctionne en-dessous de son minimum donné.

$$Z_9 = \sum_{r=1}^{p} \sum_{j=1}^{m} minreslevel_{r,j} \times MinResPen_r \tag{4.10}$$

4.6 Les contraintes

La première contrainte est celle de l'équilibre des stocks, avec à gauche de l'égalité le stock de la période précédente, augmenté de ce qui est produit et de ce qui est acheté pendant cette période. Cette quantité doit être égale à la demande (réelle ou dépendante) de la période courante, augmentée de ce qui restera éventuellement en stock. Il faut dissocier la première période des autres périodes pour tenir compte du stock initial.

$$\forall a = 1..n \in Articles :$$

$$StockInitial_a + proclevel_{a,1} + prodlevel_{a,1} = stocklevel_{a,1} + satdem_{a,1} + depdem_{a,1} \tag{4.11}$$

$$\forall a = 1..n \in Articles, \forall j = 2..m \in Jours :$$

$$stocklevel_{a,j-1} + proclevel_{a,j} + prodlevel_{a,j} = stocklevel_{a,j} + satdem_{a,j} + depdem_{a,j} \tag{4.12}$$

Il faut également qu'en toute période, le stock ne dépasse jamais la capacité de stockage de l'article :

$$\forall a = 1..n \in Articles, \forall j = 1..m \in Jours :$$

$$stocklevel_{a,j} \leq Stockmaxi_a \tag{4.13}$$

En cas de franchissement du safety stock vers le bas et uniquement vers le bas, il faudra tenir compte d'une pénalité :

$$\forall a = 1..n \in Articles, \forall j = 1..m \in Jours :$$

$$stocklevel_{a,j} - Safetystocklevel_a = \alpha_{a,j} - \beta_{a,j} \tag{4.14}$$

Il faut respecter l'équilibre de la demande. Une demande sera soit satisfaite, soit non satisfaite :

$$\forall a = 1..n \in Articles, \forall j = 1..m \in Jours :$$

$$Demand_{a,j} = satdem_{a,j} + unsatdem_{a,j} \tag{4.15}$$

Si les quantités achetées ne sont pas au moins égales à celles provenant de SNP, il faudra tenir compte d'une pénalité :

$$\forall a = 1..n \in Articles :$$

$$\sum_{j=1}^{m} proclevel_{a,j} - ProcSNP_a = \gamma_a - \delta_a \tag{4.16}$$

Le niveau de production d'un article est calculé en multipliant le taux d'utilisation de tous les PPMs d'un article par la quantité produite de cet article dans ce PPM :

$$\forall a = 1..n \in Articles, \forall j = 1..m \in Jours :$$

$$prodlevel_{a,j} = \sum_{b=1}^{o} ppmlevel_{b,j} \times PPMOut_{b,a} \tag{4.17}$$

Le respect de la nomenclature se fait par cette contrainte :

$$\forall a = 1..n \in Articles, \forall j = 1..m \in Jours :$$

$$depdem_{a,j} = \sum_{b=1}^{o} ppmlevel_{b,j} \times PPMIn_{b,a} \tag{4.18}$$

Le niveau d'utilisation d'une ressource est la somme de son utilisation par tous les PPM qui l'utilise :

$$\forall r = 1..p \in Ressources, \forall j = 1..m \in Jours :$$

$$reslevel_{r,j} = \sum_{b=1}^{o} ppmlevel_{b,j} \times PPMCapa_{b,r} \tag{4.19}$$

Chaque ressource a un maximum de production. Ce maximum est diminué des périodes d'incapacité de fonctionnement :

$$\forall r = 1..p \in Ressources, \forall j = 1..m \in Jours :$$
$$reslevel_{r,j} \leq MaxResCap_r - ResInc_{r,j} \tag{4.20}$$

Les variations des niveaux de production ne peuvent dépasser un certain niveau vers le haut. Il faut dissocier la première période des autres périodes pour tenir compte du niveau initial de la ressource.

$$\forall r = 1..p \in Ressources :$$
$$reslevel_{r,1} < ResLevIn_r \times (1 + \frac{MaxVar_r}{100}) \tag{4.21}$$

$$\forall r = 1..p \in Ressources, \forall j = 2..m \in Jours :$$
$$reslevel_{r,j} < reslevel_{r,j-1} \times (1 + \frac{MaxVar_r}{100}) \tag{4.22}$$

Les variations des niveaux de production ne peuvent pas descendre en-dessous d'un certain niveau. Il faut dissocier la première période des autres périodes pour tenir compte du niveau initial de la ressource.

$$\forall r = 1..p \in Ressources :$$
$$reslevel_{r,1} > ResLevIn_r \times (1 + \frac{MaxVar_r}{100}) \tag{4.23}$$

$$\forall r = 1..p \in Ressources, \forall j = 2..m \in Jours :$$
$$reslevel_{r,j} > reslevel_{r,j-1} \times (1 + \frac{MaxVar_r}{100}) \tag{4.24}$$

Il faut respecter la taille des lots à acheter. Cette contrainte transforme notre problème linéaire en problème linéaire en nombres entiers et sera donc exécutée ou non en fonction du fichier de paramétrage de l'optimiseur.

$$\forall a = 1..n \in Articles, \forall j = 1..m \in Jours :$$
$$proclevel_{a,j} = numberofbatches_{a,j} * ProcBatchSize_a - minbatchlevel_{a,j} \tag{4.25}$$

Nous rajoutons ici une contrainte pour nous assurer que la pénalité *minbatchlevel* soit toujours plus petite que *ProcBatchSize*. En effet, les solveurs implémentant l'algorithme "Branch Bound" peuvent s'arrêter de quatre manières différentes :

- Après un temps donné ;
- Après un nombre d'itérations donné ;
- Lorsque la solution en cours est à un pourcentage donné de la solution optimale ;
- A la solution optimale ;

Dans les trois premiers cas, il n'est pas prouvé que la pénalité *minbatchlevel* soit plus petite que *ProcBatchSize*.

$$\forall a = 1..n \in Articles, \forall j = 1..m \in Jours :$$
$$minbatchlevel_{a,j} < ProcBatchSize_a \tag{4.26}$$

La quantité minimum de produit à acheter. Cette contrainte est non linéaire vu que l'ensemble solution est non convexe. Pour résoudre cette contrainte, deux solutions ont été étudiées.
La première consiste à résoudre deux problèmes indépendants. Le premier ayant comme contrainte que les achats sur l'article a doivent être nuls, le deuxième ayant comme contrainte que les achats sur l'article a doivent être au minimum de $MinProc_a - minprocpen_{a,j}$. Cette solution impose de devoir résoudre 2^n problèmes, n étant le nombre de conditions avec un "ou" et devient vite insurmontable.

La deuxième solution, qui a été retenue, se base sur le système de contraintes suivant :
Soit il n'y a pas d'achat...

$$\forall a = 1..n \in Articles, \forall j = 1..m \in Jours : \\ proclevel_{a,j} \leq M \times x_{a,j} \tag{4.27}$$

... soit un minimum donné.

$$\forall a = 1..n \in Articles, \forall j = 1..m \in Jours : \\ proclevel_{a,j} \geq MinProc_a + M \times (x_{a,j} - 1) - minproclevel_{a,j} \tag{4.28}$$

Cette contrainte transformant notre problème linéaire en un problème linéaire en nombre entiers, l'utilisateur pourra choisir de l'exécuter ou non dans le fichier de paramétrage de l'optimiseur.

Le minimum de production des resources : à nouveau, il s'agit ici d'un minimum semi-continu et donc d'une contrainte non-linéaire. Soit la ressource ne fonctionne pas, soit elle fonctionne au-dessus d'un certain minimum. Le même système que pour la contrainte sur la quantité minimum à acheter sera employé.

Soit une ressource ne fonctionne pas...

$$\forall r = 1..p \in Ressources, \forall j = 1..m \in Jours : \\ reslevel_{r,j} \leq G \times y_{r,j} \tag{4.29}$$

... soit elle tourne au moins sur un minimum donné.

$$\forall r = 1..p \in Ressources, \forall j = 1..m \in Jours : \\ reslevel_{r,j} \geq MinResCap_r + G \times (y_{r,j} - 1) - minreslevel_{r,j} \tag{4.30}$$

Cette contrainte étant grande consommatrice en temps machine, l'utilisateur pourra choisir de l'exécuter ou non dans le fichier de paramétrage de l'optimiseur.

Chapitre 5

Le choix de l'outil informatique

La résolution d'un problème de programmation linéaire s'effectue en deux grandes phases : la modélisation et la résolution. Historiquement, les premiers logiciels ne servaient qu'à résoudre des programmes linéaires déjà modélisés et donnés sous forme numérique. Il s'agissait donc de codes algorithmiques de résolution appelés aussi solveurs. Par la suite sont venus se greffer les langages de modélisation ou modeleurs.

Pour nous aider à choisir l'outil le mieux adapté, nous avons repris certains des critères de Fourer (Fourer 2005) :

- Le prix : dans le cadre de ce projet, il est indispensable de trouver un outil qui soit gratuit. En effet, Solvay ne voulait pas débloquer de budget pour ce projet.
- La taille des problèmes : il faut un outil qui soit capable de résoudre un problèmes ayant plus de 3000 contraintes (± 17 contraintes sur ± 30 jours avec ± 6 produits) ;
- L'offre : nous préférons utiliser un outil donnant une offre complète comprenant un solveur et un modeleur.
- Le système d'exploitation : nous préferons utiliser un outil qui permet de travailler aussi bien sous Unix/linux que sous Windows. En effet, si aujourd'hui le module fonctionne sous Windows, il est destiné à être installé sur un serveur fonctionnant avec Unix.
- Les algorithmes : l'outil devra permettre le choix entre le Simplex primal, dual et la méthode des points intérieurs. Il devra également permettre de résoudre un problème linéaire en nombres entiers. Si d'autres techniques sont présentes, cela sera considéré comme un plus ;
- Le type de problème : l'outil choisi devra permettre la résolution de problème linéaire, et de problème linéaire en nombres entiers.

Comme il est utile de calculer les intervalles dans lesquels peuvent varier les coefficients de coûts c_j ou les seconds membres b_i sans que l'optimum change, il serait intéressant que l'outil puisse nous fournir une analyse de sensibilité.

5.1 La modélisation

Différentes méthodes existent pour modéliser un problème linéaire sous forme numérique. Nous avons retenu les plus utilisées : les formats MPS, CPLEX et GNU MathProg.
Pour illustrer les différents modeleurs, nous nous sommes basés sur le problème linéaire en nombres entiers suivant (Verbois, 2004) :
Un atelier mécanique cherche à recruter des mécaniciens-monteurs pour démarrer la production d'un nouveau moteur de motocyclette. Une petite annonce dans les quotidiens a permis de susciter la canditature de 2 mécaniciens-monteurs experts (catégorie C1), de 6 mécaniciens-monteurs confirmés (catégorie C2) et de 9 apprentis (catégorie C3). Les informations concernant ces candidats sont données dans le tableau suivant :

Catégorie	Nombre de moteurs par jours	Salaire quotidien (en $)	Nombre minimal d'années d'expér.
C1	20	200	10
C2	15	155	6
C3	8	90	2

L'atelier fonctionne 5 jours par semaine. La direction souhaite fabriquer le plus de moteurs possible tout en ne déboursant pas plus de 8.000 $ hebdomadaire en salaires. De plus, elle exige que le nombre total des années d'expérience de cette nouvelle main d'oeuvre soit d'au moins 60 ans.

5.1.1 La matrice MPS

Le format MPS, développé dans les années 1960 par IBM, est devenu le standard *de facto* accepté par la plupart des solveurs commerciaux (Mosek, Visual Xpress, OMP, ...) ou non (GLPK, COIN-OR,...). Dans les années 60, les fichiers étaient des cartes perforées. Le format MPS est donc orienté en colonnes (le modèle n'est donc pas entré en équation). Les champs doivent commencer en colonnes 1, 5, 15, 25, 40 et 50. Les différentes sections d'un fichier MPS sont séparées par un en-tête (NAMES, ROWS, COLUMNS, RHS, BOUNDS ou ENDATA) qui commence toujours à la colonne 1. Les noms choisis pour les contraintes et les variables n'ont pas d'importance pour le solveur. On essayera donc de prendre les noms les plus parlants possible.
Le problème écrit au format MPS donne :

```
NAME          ATELIER
ROWS
 N  MOTEUR
```

```
 L  CONT1
 G  CONT2
COLUMNS
    X         MOTEUR          100     CONT1           200
    X         CONT2           10
    Y         MOTEUR          75      CONT1           155
    Y         CONT2           6
    W         MOTEUR          40      CONT1           90
    W         CONT2           2
RHS
    RHS1      CONT1           1600    CONT2           60
    RHS1      MYEQN              7
BOUNDS
 UI BND1      X               2
 UI BND1      Y               6
 UI BND1      W               9
ENDATA
```

La première section NAME doit commencer à la colonne 15 et représente le nom du problème. Il n'y a pas de limitation dans le nom que l'on veut donner.
La section ROWS définit le nom de toutes les contraintes. Les lettres en colonne 2 et 3 donnent la direction de l'équation : E pour "égale", L pour "plus petit", G pour "plus grand" et N pour la fonction "objectif".
C'est dans la section COLUMNS que l'on place toute les entrées de la matrice A. Toutes les entrées d'une variable doivent se suivre. Si pour une équation, une variable n'a pas d'entrée dans la section COLUMNS, elle est supposée avoir un coefficient nul.
La section RHS nous permet de définir le vecteur de droite des équations. Les variables non mentionnées dans le vecteur seront supposées nulles.
La section optionnelle BOUNDS nous permet de définir des limites supérieures et inférieures pour les variables. Le mot clé UP est utilisé pour les limites supérieures, UI pour les limites supérieures d'une variable en nombres entiers, LO pour les limites inférieures et LI pour les limites inférieures en nombres entiers. Les variables non mentionnées dans BOUNDS sont supposées être strictement positives et sans limite supérieure.
Le fichier doit se terminer par ENDATA. Il n'y a rien dans le fichier MPS qui donne la direction de l'optimisation. Cependant, une règle implicite est utilisée par défaut : la première ligne de type N de la section ROWS est considérée comme la fonction "objectif" et sera minimisée.

5.1.2 CPLEX LP

Le format CPLEX LP a été développé à la fin des années 1980 par CPLEX optimization, Inc. comme format d'entrée pour le solveur CPLEX linear programming system. Depuis, il a été racheté par Ilog. Le format CPLEX est beaucoup moins utilisé que le format MPS mais il est beaucoup plus facile car construit en format orienté ligne.
Un fichier au format CPLEX se divise en 5 sections :

- Définition de la fonction "objectif" : cette section doit être la première du fichier CPLEX. Elle définit la fonction "objectif" et spécifie la direction de l'optimisation. La définition de la fonction "objectif" aura la forme suivante :

$$\left\{ \begin{array}{l} minimize \\ maximize \end{array} \right\} f : scxscx...scx$$

  où f est un nom symbolique pour la fonction "objectif" ;
  s est le signe + ou - ;
  c est une constante numérique qui représente le coût ou le profit associé à une variable ;
  x est une variable.

- Définition des contraintes : la section dans lesquelles nous définirons les contraintes doit suivre la définition de la fonction "objectif". Cette section doit se présenter sous la forme suivante :
  subect to
  $constraint_1$
  $constraint_2$
  ...
  $constraint_n$
  Chaque contrainte doit commencer sur une nouvelle ligne mais peut prendre plusieurs lignes. La forme des contraintes est la suivante :

$$r : saxsax...sax \left\{ \begin{array}{l} <= \\ >= \\ = \end{array} \right\} b$$

  où r est un nom symbolique pour la contrainte ;
  s est le signe + ou - ;
  a est une constante numérique ;
  x est une variable ;
  b est le second membre constant de la contrainte.

- Définition des bornes : toutes les contraintes exprimant une borne inférieure ou supérieure sont déclarées dans la section non obligatoire

commençant par le mot-clé "bounds". Chaque borne doit avoir une des formes suivantes :
$x >= l$ borne inférieure
$l <= x$ borne inférieure
$x <= u$ borne supérieure
$u >= x$ borne supérieure
$x = t$ variable fixée
x free variable libre
où x est une variable ;
l est une constante éventuellement signée qui définit la borne inférieure de la variable ou qui peut prendre la valeur "-inf" ce qui signifie alors que la variable n'a pas de borne inférieure ;
u est une constante éventuellement signée qui définit la borne supérieure de la variable ou qui peut prendre la valeur "+inf" ce qui signifie alors que la variable n'a pas de borne supérieure ;
t est une constante éventuellement signée qui fixe la valeur d'une variable.

- Type de problème : cette section non obligatoire permet de spécifier si certaines variables sont booléennes ou en nombres entiers. Elle est de la forme :

$$\left\{ \begin{array}{l} general \\ integer \\ binary \end{array} \right\}$$

x_1
x_2
...
x_n
où x_k est le nom d'une variable, $k = 1, ..., n$.

- Fin de fichier : le fichier CPLEX doit obligatoirement se terminer par le mot-clé "End".

L'écriture de notre problème en CPLEX LP donnerait :

```
Maximize  MOTEUR : 100 X + 75 Y + 40 W
Subject To
200 X + 155 Y + 90 W <= 1600
10 X + 6 Y + 2 W >= 60
Bounds
0 <= X <= 2
0 <= Y <= 6
```

```
0 <= W <= 9
Integer
X
Y
W
End
```

5.1.3 GNU MathProg

GNU MathProg est un sous-ensemble de AMPL. AMPL est le modeleur le plus connu pour les problèmes d'optimisation linéaire et non linéaire. Il a été développé par les laboratoires Bell. Sa particularité est de pouvoir appeler pratiquement tous les solveurs existants. (Fourrer et ass., 2002).
Nous avons choisi d'utiliser GNUMathProg car il est le modeleur gratuit le plus puissant que nous ayons trouvé. De plus il était le plus facile à utiliser. Les données du problème changeant à chaque exécution, il faut un outil qui permette de les mettre à jour facilement. Les notations mathématiques constituent un moyen facile pour écrire les expressions de manière concise et générale :

Ensembles

- N = nombre de catégories de mécaniciens-monteurs(i=1,...,N)

Paramètres

- C_i = nombre de candidats nécaniciens-monteurs dans la catégorie i
- M_i = nombre de moteurs montés par jour par les mécaniciens-monteurs de la catégorie i(en unités)
- S_i = salaire quotidien des mécaniciens-monteurs de la catégorie i (en \$)
- E_i = nombre minimal d'années d'expérience des mécaniciens-monteurs de la catégorie i (en années)
- CSM = coût salarial hebdomadaire maximum (en \$)
- AEM = nombre minimum d'années d'expérience de l'ensemble mécaniciens-monteurs engagés (en années)
- NJS = nombre de jours de travail par semaine (en jours)

Variables

- z = nombre de moteurs fabriqués par jour
- x_i = nombre de mécaniciens monteurs de la catégorie i engagés

Objectif

- $Z = Max \sum_{i=1}^{N} M_i \times x_i$

Contraintes

- $NJS \times \sum_{i=1}^{N} S_i \times x_i \leq CSM$
- $\sum_{i=1}^{N} E_i \times x_i \geq AEM$
- $x_i \leq C_i$ pour $i = 1, ..., N$
- $x_i \geq 0$ et entiers pour $i = 1, ..., N$

Avec GnuMathProg, il est possible d'écrire une description compacte du problème, que nous appellerons *modèle* en utilisant des notations algébriques pour l'objectif et les contraintes.
Cinq composants de base sont à distinguer :

- Les *ensembles* comme les catégories de mécaniciens-monteurs. Ils sont déclarés par le mot-clé "set".
- Les *paramètres* comme le nombre de moteurs montés par jour par les mécaniciens-monteurs d'une catégorie donnée. Ils sont déclarés par le mot-clé "param".
- Les *variables* dont la valeur sera déterminée par le solveur. Elles sont déclarées par le mot-clé "var".
- L'*objectif* à maximiser ou à minimiser. Il est déclaré par le mot-clé "maximize" ou "minimize".
- Les *contraintes* que la solution doit satisfaire. Elles sont déclarées par le mot-clé "s.t."

L'exemple devient donc en GNUMathProg :

```
set N;

param C{i in N};
param M{i in N};
param S{i in N};
param E{i in N};
param CSM;
param AEM;
param NJS;

var z;
var x{i in N}, integer, >= 0;

maximize Moteur:sum{i in N} M[i] * x[i];
```

```
s.t. cont1: NJS * sum{i in N} S[i] * x[i] <= CSM;
s.t. cont2: sum{i in N} E[i] * x[i] >= AEM;
s.t. cont3{i in N}: x[i] <= C[i];
```

Le modèle décrit un nombre infini de problèmes d'optimisation apparentés. Cependant, si nous spécifions des valeurs spécifiques aux données, le modèle devient un problème spécifique c'est-à-dire une *instance* de ce modèle qui peut être résolue. Il reste donc à écrire le fichier des données :

```
data;
set N:= C1 C2 C3;
param C:= C1    2
          C2    6
          C3    9;
param M:= C1    20
          C2    15
          C3    8;
param S:= C1    200
          C2    155
          C3    90;
param E:= C1    10
          C2    6
          C3    2;
param CSM 8000;
param AEM 60;
param NJS 5;
end;
```

5.2 La résolution

Deux solveurs gratuits ont été étudiés : GLPK et COIN-OR.

5.2.1 COIN-OR

Le projet COIN-OR (COmputational INfrastructure for Operations Research) est une initiative pour promouvoir le développement de logiciels gratuits dans la communauté de la recherche opérationnelle. Mais COIN-OR est également un réseau de chercheurs académiques et industriels décidés à améliorer l'informatisation de la recherche opérationnelle (Lougee, 2003). La librairie COIN est composée entre autres des modules suivants :

- OSI (Open Solver Interface) est une interface de programmation qui nous permet de lier les différents solveurs de COIN avec différents modeleurs. Elle permet de résoudre des problèmes modélisés à l'aide de MOSEK, ILOG CPLEX, XPRESS-MP ou encore au format MPS. Il est à noter que COIN-OR n'est fourni avec aucun de ces modeleurs.
- CLP (Coin-OR LP) est un solveur qui implémente l'algorithme du Simplex primal et dual. Il permet également d'utiliser les critères de Dantzig permettant de minimiser le nombre d'itérations effectuées au cours du déroulement de l'algorithme. Il permet de résoudre des problèmes jusqu'à 1,5 millions de contraintes.
- CGL (Cut Generator Library) est un ensemble de méthodes permettant de générer des "cuts" valides dans des problèmes linéaires en nombre entier.
- BCP (Branch, Cut and Price Library) est un ensemble de méthodes Branch & Cut dévelopées par Ladany (Ladany 1996) en C++. Le problème relaxé peut-être résolu par n'importe quel solveur linéaire compatible avec l'interface OSI.
- CBC (COIN-OR Branch and Cut) est une librairie implémentant la méthode Branch & Cut. Elle a été dévelopée à l'origine par John Forrest pour tester CLP.
- IPOPT (Interior Point Optimization) implémente la méthode des points intérieurs.
- DFO (Derivative Free Optimization) est utilisé pour résoudre des problèmes d'optimisation non-linéaires.
- OTS (Open Tabu Search) implémente la recherche Tabou.
- SYMPHONY est une librairie appelable pour la résolution de problème linéaires en nombres entiers (ou mixte) très proche de BCP. Le problème relaxé peut-être résolu par n'importe quel solveur linéaire compatible avec l'interface OSI.

L'avantage de la librairie COIN est qu'il permet de changer le moteur d'optimisation facilement.

5.2.2 GLPK

GLPK (GNU Linear Programming Kit) est une bibliothèque de sous-programmes écrits en ANSI C et organisée sous forme de librairie appelable. Cet outil est utilisé pour résoudre des problèmes de grande taille (jusqu'à 100.000 contraintes) en programmation linéaire et des problèmes de plus petite taille (100 à 200 variables) en programmation linéaire en nombre entier. Il a été développé par Andrew Makhorin (Moscow Aviation Institute, Moscou, Russie). La bibliothèque GLPK inclut les composants suivants :

- l'implémentation de l'algorithme du simplex primal et dual ;
- l'implémentation de la méthode de points intérieurs (primal et dual) ;

- l'implémentation de la méthode Branch & Bound. La méthode Branch & Bound fournie avec GLPK ne peut résoudre le problème linéaire relaxé qu'avec le Simplex fournit par GLPK. Nous pouvons paramétrer la méthode pour qu'elle effectue ses recherches en profondeur d'abord ou en largeur d'abord ;
- le langage de modélisation mathématique GNU MathProg ;
- une interface de programmation qui permet de travailler au format MPS, CPLEX ou GNU MathProg.

Le principal avantage de GLPK est que la formulation est facile à comprendre et à maintenir. Son désavantage est de compliquer un changement éventuel dans le moteur d'optimisation. Nous avons retenu le solveur GLPK car, contrairement à COIN-OR, il incluait un excellent langage de modélisation gratuit et qu'il fonctionnait tout aussi facilement sous windows que sous linux. De plus son utilisation dans le cadre de ce projet, nous a montré qu'il fonctionnait bien.

Chapitre 6

Intégration de l'optimiseur

L'optimiseur défini plus haut devra s'intégrer au sein du système informatique de Solvay. Nous devrons donc extraire les données nécessaires de SAP/APO, les envoyer à l'optimiseur et enfin présenter les résultats à l'utilisateur.

Nous avons choisi le standard XML pour interfacer notre optimiseur avec APO. XML (Extensible Markup Language ou langage de balisage extensible) est le standard du World Wide Web Consortium (W3C) pour la représentation générique de données structurées. Le W3C est une organisation internationale qui regroupe plus de 500 membres originaires de 34 pays. Parmi eux, on compte des instituts de recherche (CERN, NASA, ...), des organisations gouvernementales (NATO Consultation, National Security Agency, ...) et de nombreux groupes industriels (IBM, Nokia, ...). Le W3C a été fondé en 1994 par Tim Berners-Lee. Il a pour rôle l'émission de spécifications concernant les technologies gravitant autour de l'Internet. Le W3C est notamment à l'origine du langage HTML ou du protocole HTTP. Du fait de son indépendance et de son impartialité, le W3C a fortement contribué à l'interopérabilité du Web par la création et la promotion de langages et protocoles non propriétaires. Ceci est rendu possible par la pratique du consensus entre les membres du consortium et de forums de discussion ouverts à tous.

XML est une technologie universelle, particulièrement adaptée aux environnements informatiques hétérogènes.

En apportant un standard ouvert, portable, facile à mettre en oeuvre, à la fois souple et structurant, de surcroît gratuit et libre de droits, adapté au web et indépendant des plates-formes, des langages et des systèmes d'exploitation, XML s'impose comme support privilégié de l'information.

La séparation des données et de la présentation, le format des données orienté objet, les nombreuses technologies associées telles que la mise en forme et la validation syntaxique, la portabilité sur tous les langages de programmation sont les atouts qui font qu'XML n'est pas une simple mode, mais bien une technologie qui a sa place dans les applications d'aujourd'hui

et de demain.

Un des buts d'XML est de faire communiquer des programmes écrits en différents langages. En effet, XML est un standard au format texte ce qui en fait un formidable format d'échange d'informations. Ainsi, deux applications différentes sont capables de communiquer entre elles via un document XML.

Le langage XML est basé sur l'utilisation et la création de balises. Toutes les informations contenues dans un document XML sont encadrées par les balises XML. Les balises nous renseignent sur la hiérarchie et la sémantique des données. Un document XML est composé de deux éléments : d'une part, la structure des données et, d'autre part, les données elle-mêmes. La balise qui fait partie de la structure des données représente le contenant et l'intérieur de ces balises le contenu.

Nous aurons trois fichiers XML différents en entrée de l'optimiseur, un pour les données maîtres, un pour les données transactionnelles et le dernier sera le fichier de paramétrisation de l'optimiseur (cfr. infra). Le fichier des données maîtres sera décrit sous la forme d'une structure arborescente autodescriptive. Les noeuds de l'arbre correspondront aux tables des données maîtres et les feuilles aux contenus de ces tables.

Le fichier des données transactionnelles sera lui aussi structuré en fonction des tables des données maîtres mais ne contiendra que les données transactionnelles.

Le fichier de paramétrisation contiendra une feuille par contrainte non-obligatoire (taille des lots, production minimum et achat minimum). la valeur de la feuille sera "1" si la contrainte doit être respectée, "0" sinon.

Pour que la communication entre SAP et l'optimiseur soit possible, il faut que l'optimiseur se dote d'un dispositif lui permettant de manipuler un document XML pour exploiter ses données. Ce dispositif est appelé un "parser".

Le "parser" devra d'abord s'assurer que le document XML est bien formé. Un document XML est bien formé lorsqu'il est syntaxiquement correct (toutes les balises doivent être fermées et les balises ne peuvent être imbriquées ou croisées). Ensuite, le "parser" réalise un arbre syntaxique du document XML. Le "parser" étant un utilitaire de bas niveau, le résultat du parsing n'est pas visible. C'est pourquoi, il existe des interfaces de programmation (API Appplication Programming Interface) qui proposent des services facilitant la manipulation du document XML. On distingue deux types d'API pour XML :

- DOM (Document Object Model), l'API standardisée par le W3C ;
- SAX (Simple API for XML) qui a été développé sur le modèle des logiciels libres et qui est un standard *de facto*.

Un "parser" DOM compile le fichier XML en mémoire sous forme d'un arbre d'objets. Une fois l'arbre créé, il est manipulable grâce à des méthodes d'accès en consultation et en modification fournies par l'API DOM. Le par-

sing DOM aboutit au chargement en mémoire d'un arbre d'objets représentant le document XML parsé. Ce mécanisme est coûteux en temps et en performance.

Le "parser" SAX ne compile pas le document XML dans un arbre chargé en mémoire, mais il émet des évenements de parsing tel que l'ouverture ou la fermeture d'une balise. Ces événements sont utilisés par l'application pour effectuer des traitements. L'API SAX met à la disposition de l'application un nombre fini de méthodes d'accès au document. Ces méthodes correspondent aux événements que le "parser" est suceptible d'intercepter au cours de sa lecture. SAX permet de parcourir le document XML sans le conserver en mémoire.

Nous avons choisi un parser SAX vu que le traitement de l'information est plus rapide qu'avec DOM puisque la phase de stockage en mémoire est inexistante.

Nous avons choisi le parser Expat© développé par James Clark car il est gratuit et disponible sous la licence libre de droit du MIT. Expat est un parser de type Sax écrit en langage C. Il fallait que le parser utilisé soit compatible avec le langage C vu qu'il devra s'intégrer dans le module d'optimisation lui-même écrit en C. De plus, Expat est très rapide (Cooper, 1999).

En sortie, l'optimiseur fournira un fichier XML contenant l'ensemble des résultats. Ce fichier sera mis en forme à l'aide d'une feuille de style XSL. XSL *(eXtensible Style sheet Language)* est la technologie de feuilles de styles associées à XML. Contrairement à l'HTML dans lequel les données et leurs informations de présentation sont mélangées dans le même fichier, le consortium W3C a décidé de permettre de séparer le fond et la forme d'un document XML à l'aide des feuilles de styles.

```xml
<?xml version="1.0"?>
<master_data nom="nom de l'unité de fabrication">
    <!-- Les produits -->
    <produit id="Nom du produit 1">
        <ssl>Niveau du Stock de sécurité</ssl>
        <stockmax>Niveau du stock maximum</stockmax>
        <storcost>Coût de stockage</storcost>
        <ssc>Pénalité en cas de franchissment du safety stock</ssc>
        <nondelpen>Pénalité en cas de demande non satisfaite</nondelpen>
        <proccost>Coût d'achat du produit 1</proccost>
    </produit>
    ...
    <produit id="Nom du produit n">
        ...
    </produit>


    <!-- Les ressources -->
    <ressource id="Nom de la ressource 1">
        <maxrescap>Capacité maximum de production</maxrescap>
        <maxvar>Variation maximum de production</maxvar>
    </ressource>
    ...
    <ressource id="Nom de la ressource n">
        ...
    </ressource>


    <!-- Les ppms -->
    <ppm nom="Nom de la PPM1">
        <cost>Coût d'utilisation de la PPM1</cost>
        <ppmin id="mp 1">Quantitée nécessaire de matière première 1</ppmin>
        <ppmin id="mp 2">Quantitée nécessaire de matière première 2</ppmin>
        <ppmout prod="pf 1">Quantité produite du produit fini 1</ppmout>
        <ppmout prod="pf 2">Quantité produite du produit fini 2</ppmout>
        <ppmcapa ress="Ress">Temps d'utilisation de la ressource</ppmcapa>
    </ppm>
    ...
    <ppm nom="Nom de la PPMn">
        ...
    </ppm>
</master_data>
```

FIG. 6.1 – Fichier XML pour les données maîtres

```xml
<?xml version="1.0"?>
<transactional_data nom="nom de l'unité de fabrication">
    <!-- Les produits -->
    <produit id="Nom du produit 1">
        <stockinit>Niveau du stock initial</stockinit>
        <dem j="1">La demande au jour 1</dem>
        ...
        <dem j="n">La demande au jour n</dem>
    </produit>
    ...
    <produit id="Nom du produit n">
        ...
    </produit>

    <!-- Les ressources -->
    <ressource id="Nom de la ressource 1">
        <reslevin>Niveau inital de production</reslevin>
    </ressource>
    ...
    <ressource id="Nom de la ressource n">
        ...
    </ressource>

</transactional_data>
```

FIG. 6.2 – Fichier XML pour les données transactionnelles

```xml
<?xml version="1.0" ?>
- <param_data nom="VC">
    <!-- Les contraintes -->
    <b_minproc>0</b_minproc>
    <b_minprod>0</b_minprod>
    <b_batchsize>0</b_batchsize>
  </param_data>
```

FIG. 6.3 – Fichier XML pour la paramétrisation

Chapitre 7

Principes de programmation

Le but poursuivi par la création d'un module d'optimisation de production est d'offrir à l'utilisateur un outil s'adaptant à tous types de production et fournissant un plan de production tenant compte de plusieurs contraintes.

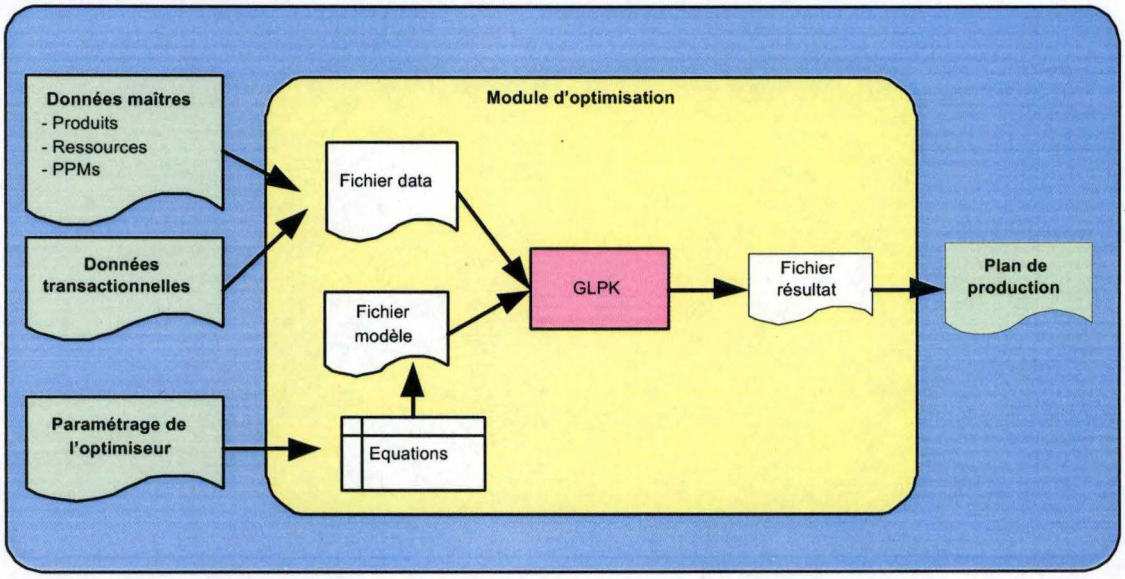

FIG. 7.1 – Modélisation du module d'optimisation

Pour permettre l'optimisation, un analyste introduira une série de données (les données maîtres) modélisant une production dans un fichier XML. Il spécifiera également dans un autre fichier XML (paramétrage de l'optimiseur) quelles sont les contraintes qui sont d'applications dans le cadre de la production modélisée. L'utilisateur final, quant à lui, introduira les données journalières (données transactionnelles) de production dans un dernier fichier XML. Notre module fournira alors un plan de production optimum.

A terme, les fichiers XML seront transmis par le système informatique SAP, c'est pourquoi la structure de nos fichiers XML respectent la structure de données du système SAP.

7.1 Données maîtres

On appelle données maîtres les données qui varient peu dans le temps. Elles regroupent les contraintes liées intrinsèquement aux types de production, aux produits et aux équipements utilisés dans la production. Ce sont ces données qui permettent de modéliser une production. Les données maîtres impliquées dans notre modèle sont les suivantes :

Données maîtres pour les produits

- Stock maximum du produit
- Coût de stockage
- Niveau du stock de sécurité
- Pénalités associées au dépassement du stock de sécurité
- Pénalités en cas de non livraison
- Coût d'achat
- Pénalités en cas de non respect du minimum à acheter
- Pénalités en cas de non respect des ordres d'achats obligatoires
- Pénalités en cas de non respect de la taille des lots à acheter

Données maîtres pour les PPMS

- Coût d'utilisation du mode opératoire
- Quantité pour chacune des matières premières nécessaires au mode opératoire (PPMin)
- Quantité de chacun des produits finis fabriqués par le mode opératoire (PPMout)
- Rendement de chaque mode opératoire (PPMcapa) pour une ressource spécifiée

Données maîtres pour les ressources

- Maximum de production d'une ressource
- Minimum de production d'une ressource
- Pénalités en cas de franchissement du minimum
- Variation maximum des niveaux de production

Le lecteur trouvera en annexe des exemples de fichiers XML contenant les données maîtres.

7.2 Données transactionnelles

On appelle données transactionnelles les données qui varient d'un plan de production à l'autre. Les données transactionnelles impliquées dans notre modèle sont les suivantes :

Données transactionnelles pour les produits

- Stock initial
- Achats obligatoires
- Quantité minimum à acheter
- Quantités demandées pour chaque jour de l'horizon

Données transactionnelles pour les ressources

- Niveau d'utilisation initial des ressources
- Incapacité de production pour chaque jour de l'horizon

Le lecteur trouvera en annexe des exemples de fichiers XML contenant les données transactionnelles.

7.3 Le paramétrage de l'optimiseur

Etant donné que certaines contraintes sont de grandes consommatrices de temps machine, l'utilisateur pourra définir quelles contraintes lui sont nécessaires. Il en sera ainsi pour les contraintes utilisant des variables binaires ou entières. Ces contraintes sont également écrites dans un fichier XML et sont :

- Quantité minimum à acheter
- Quantité minimum à produire
- Taille des lots de produits à acheter

7.4 Le plan de production

En sortie, l'optimiseur fournira pour chacun des jours de l'horizon de planification :

- la quantité à produire pour chaque produit ;
- la quantité à acheter pour chaque produit ;
- les demandes qui ont pu être satisfaites, celles qui ont dû être annulées ;
- le stock à la fin de chaque jour pour chaque produit ;
- le niveau d'utilisation de chaque ressource ;
- le coût total du plan de production ;

7.5 Architecture du module

Le solveur que nous avons utilisé est le solveur **GLPK**. Pour fonctionner, ce solveur nécessite en entrée un fichier *data* et un fichier *modèle*. Le fichier *data* reprend l'ensemble des variables qui seront intégrées dans les équations contenues dans le fichier *modèle*.

Les équations modélisant le problème sont écrites par le module dans un fichier *modèle*. Les équations gérant les contraintes paramétrables seront ou non écrites dans ce fichier en fonction du fichier de paramétrage.

Les données maîtres et les données transactionnelles sont converties par le programme en différentes listes chaînées. Chaque variable est à l'origine d'une liste chaînée. Ces listes chaînées sont ensuite utilisées pour écrire le fichier *data*.

Les fichiers *data* et *modèle* sont ensuite intégrés dans le solveur GLPK qui fournit les solutions des équations sous forme d'un fichier texte. Le module va alors lire dans ce fichier texte les résultats qui nous intéressent et les écrire dans un fichier XML. Ce fichier XML est mis en forme à l'aide d'un fichier XSL et présente le plan de production à l'utilisateur.

L'ensemble du module est écrit en langage C.

Chapitre 8

Application à la production du chlorure de vinyle

8.1 La fabrication du VC

Le chlorure de vinyle (VC) est le monomère utilisé pour la fabrication du polychlorure de vinyle (PVC). Solvay est actuellement le deuxième plus gros fournisseur de PVC en Europe avec près d'1.300.000 tonnes produites par an. Ses sites de production sont Jemeppe et Anvers en Belgique, Rheinberg et Ludwigshafen en Allemagne, Martorell en Espagne et Tavaux en France.

Le chlore (Cl_2) obtenu à partir de l'électrolyse du sel, et l'éthylène (C_2H_4) réagissent dans des réacteurs appelés chlorateurs pour donner le dichloroéthane DCE ($C_2H_4Cl_2$).
Celui-ci, pyrolysé dans des fours au gaz naturel, c'est-à-dire chauffé au point d'en "cracker" les molécules, est décomposé en VC (C_2H_3Cl) et en acide chlorhydrique (ClH [1]).
Le VC est consommé pour la fabrication du PVC.
Selon le premier principe de la thermodynamique, le ClH sortant des fours est repris dans des réacteurs d'oxychloration où réagissent ClH, éthylène (C_2H_4) et oxygène (O_2) pour former du dichloroéthane DCE ($C_2H_4Cl_2$) et de l'eau. Il y a donc deux sources de production de dichloroéthane : l'une par chloration et l'autre par oxychloration.

Dans le cadre de ce projet, nous ne nous intéresserons qu'aux produits dont l'approvisionnement est critique et qui sont utilisés en quantités importantes lors de la fabrication. Nous appelons ces produits les articles tactiques. Ainsi, nous ne tiendrons pas compte de l'oxygène, du gaz naturel, de la vapeur,...

[1] L'acide chlorhydrique sera appellé ClH lorsqu'il est sous forme gazeuse et HCl lorsqu'il est liquide.

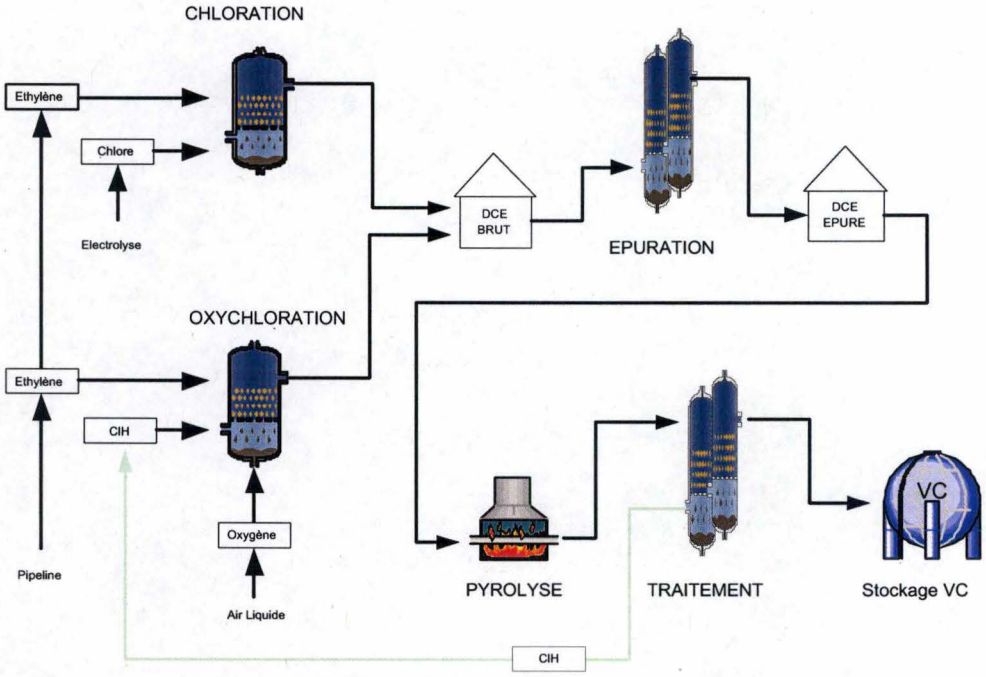

FIG. 8.1 – La production du VC à l'usine de Jemeppes

8.2 Le choix des données

Le modèle des données générales applicable à notre optimiseur est le suivant :

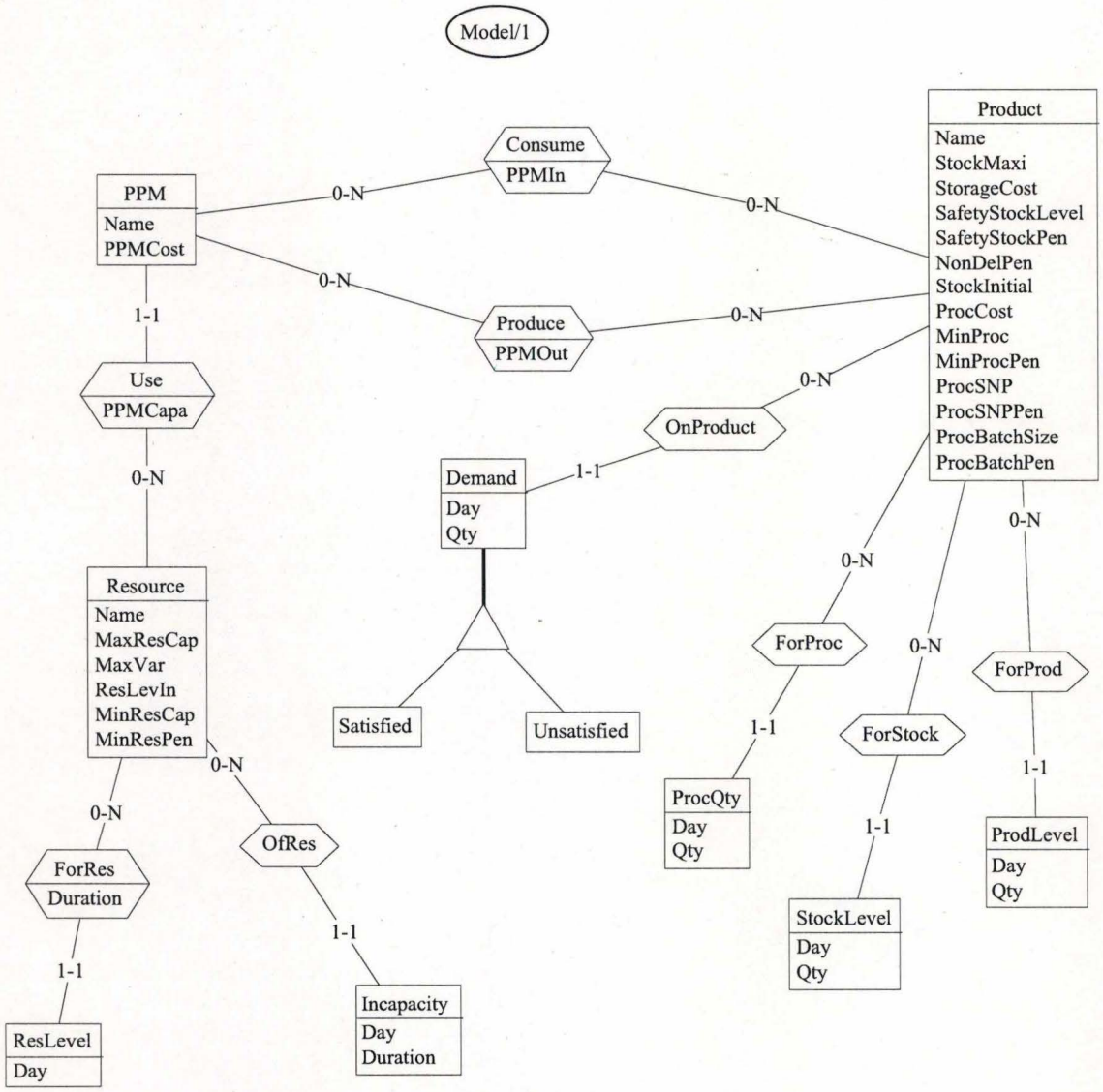

FIG. 8.2 - Modèle des données

Dans le cas particulier de la fabrication du VC, ce schéma [2] est appliqué comme présenté à la figure 8.3.

[2]Toutes les données présentées dans ce schéma ont été légèrement modifiées pour des raisons de confidentialité.

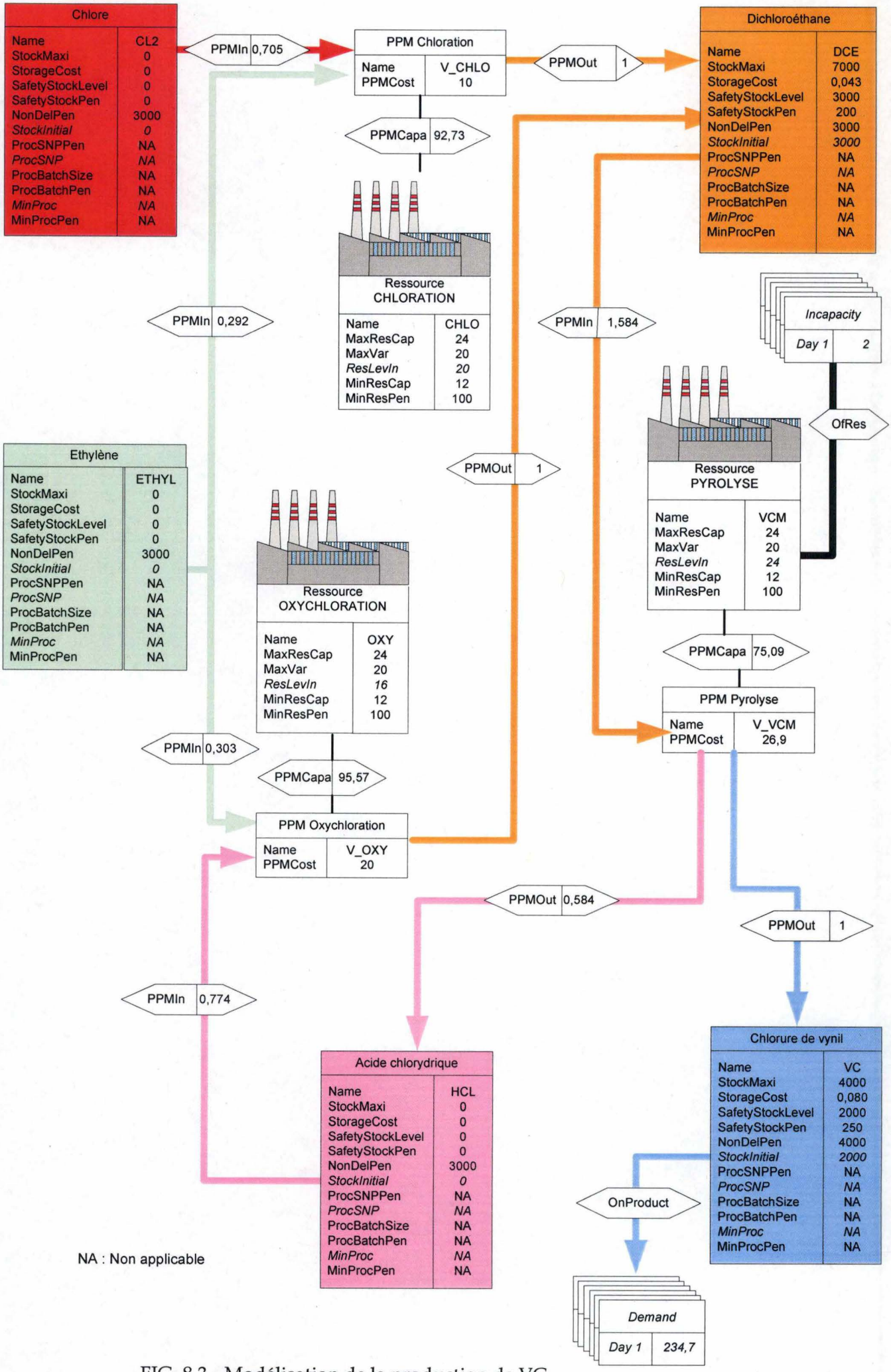

FIG. 8.3 - Modélisation de la production de VC

Nous considérerons un horizon de planification de 30 jours.

Les différents produits (Chlore, acide chlorhydrique, éthylène, dichloroéthane et VC) sont représentés par des tableaux qui reprennent les différentes caractéristiques propres à ces produits (stock maximum, safety stock, coût de stockage, ...). Une série de caractéristiques (Achats obligatoires, Taille des lots de produit à acheter, Quantité minimum à acheter, ...) ne sont pas applicables ici. En effet ces contraintes ne sont pas d'application dans le cadre de la production du VC (vu que les livraisons de matières premières se font par pipeline) elles ne doivent donc pas être prises en compte par l'optimiseur.

Comme explicité dans le chapitre 3, le mode opératoire de la fabrication est représenté par les données du PPM. Tous les PPMs sont également représentés par des tableaux qui reprennent leurs différentes caractéristiques. Ainsi, nous voyons que pour produire 1 tonne de Chlorure de vynile (PPMOut du PPM "V_VCM"), nous avons besoin d'1,584 tonne de Dichloroéthane (PPMIn du PPM "V_VCM") et que nous produirons 0,584 tonne d'acide chlorhydrique (PPMOut du PPM "V_VCM") comme co-produit. Nous voyons également que la production d'1 tonne de VC aura un coût de 26,9. De même nous voyons que pour produire 1 tonne de Dichloroéthane, nous avons la possibilité soit de consommer 0,705 tonne de Chlore et 0,292 tonne d'Ethylène à un coût de 10 (PPM "V_CHLO" qui repésente la chloration) soit de consommer 0,774 tonne d'acide chlorhydrique et 0,303 tonne d'Ethylène pour un coût de 20 (PPM "V_OXY" qui représente l'oxychloration). Enfin nous voyons également que, ni le chlore, ni l'éthylène, ne sont produits et qu'ils doivent donc être considérés comme des matières premières à acheter.

Tous nos PPMs sont attachés à une ressource dans laquelle se déroule la fabrication. Même si notre modèle général nous permet d'avoir plusieurs PPMs pour une ressource (vu qu'une ressource peut produire différents produits en diverses proportions en fonction de certains réglages), dans le cadre de la fabrication du VC nous aurons toujours un seul PPM par ressource. Chacune des ressources peut être en incapacité temporaire sur plusieurs jours de l'horizon (exprimée en heures/jour). Pour des raisons de clarté, nous n'avons représenté qu'une incapacité se produisant sur la ressource Pyrolyse.

Enfin, la demande concerne le produit fini (dans notre cas le chlorure de vinyle). Elle est indiquée en tonnes pour chaque jour de l'horizon de production.

L'ensemble des données reprises dans ce schéma est intégré dans deux fichiers XML, un pour les données maîtres et un autre pour les données transactionnelles (indiquées en italique dans le schéma).

8.3 Résultats

Après avoir introduit dans l'outil les différentes contraintes de production, l'optimiseur va fournir à l'utilisateur deux types de données : le plan de production à appliquer (quantité à acheter pour chaque produit et niveau d'utilisation de chaque ressource) et les projections établies sur base du plan de production (niveau de production, satisfaction de la demande, niveau du stock, coût total de production). Il est donc important de souligner que l'optimiseur permet à l'utilisateur d'évaluer l'efficacité du plan proposé. En d'autres termes, le logiciel est doté d'une auto-évaluation.
Il est cependant à noter que le coût total de production n'est pas comparable avec le coût réel de production. En effet, vu la difficulté de prendre en compte tous les paramètres intervenant dans le coût de production du produit fini, de nombreuses simplifications ont dû être réalisées, de telle sorte que le coût total de production calculé par l'optimiseur est potentiellement fort éloigné du coût réel de production. L'intérêt de connaître cette variable est donc avant tout de comparer la rentabilité de divers scenarii proposés à l'optimiseur. Les comparaisons ne sont donc possibles que de manière interne à l'optimiseur.

Plan de production

Le premier tableau indique à l'utilisateur les quantités de matières premières à acheter en fonction des différents jours du mois. Ce chiffre est exprimé en tonnes.

Procurement	Day 1	Day 2	Day 3	...	Day 30
CL2	475,611	380,489	409,057	...	514,874
EDC	0	0	0	...	0
HCL	0	0	0	...	0
VCM	0	0	0	...	0
ETHYL	382,930	306,344	329,345	...	414,542

TAB. 8.1 – Quantités à acheter

On constate qu'il n'y a pas d'achats nécessaires ni d'EDC, ni d'HCl, ni de VCM. Ceci est évidemment le reflet du fait que ces trois produits sont fournis par le processus de fabrication lui-même.
Le second tableau indique à l'utilisateur le temps par journée d'utilisation

des ressources de production.

Ressource level use	Day 1	Day 2	Day 3	...	Day 30
EDC_CHLO	17,37	13,90	14,94	...	18,81
EDC_OXY	12,80	10,24	11,00	...	13,85
VCM	21,59	17,27	18,56	...	23,37

TAB. 8.2 – Utilisations des ressources

On a pu constater sur base de ces résultats que l'étape limitante de la production est celle du VCM. En effet, seule cette ressource de production doit fonctionner quasiment 24 heures sur 24. Les seules périodes où l'unité de production peut ou doit fonctionner sont celles où il existe une incapacité de production ou lorsque la quantité demandée est plus faible.

Projections établies

Les deux premiers tableaux suivants fournissent à l'utilisateur les informations concernant la satisfaction de la demande et l'importance de la demande non satisfaite.

Satisfied Demand	Day 1	Day 2	Day 3	Day 4	...	Day 30
CL2	0	0	0	0	...	0
EDC	0	0	0	0	...	0
HCL	0	0	0	0	...	0
VCM	234,743	895,171	896,708	896,708	...	880,455
ETHYL	0	0	0	0	...	0

TAB. 8.3 – Demandes satisfaites

Unsatisfied Demand	Day 1	Day 2	Day 3	Day 4	...	Day 30
CL2	0	0	0	0	...	0
EDC	0	0	0	0	...	0
HCL	0	0	0	0	...	0
VCM	0	0	0	0	...	0
ETHYL	0	0	0	0	...	0

TAB. 8.4 – Demandes non satisfaites

Sur base du scénario introduit par l'utilisateur et du plan de production

proposé par l'optimiseur, on constate que toute la demande a pu être satisfaite.

Le troisième tableau fournit la quantité produite pour chaque produit.

Production	Day 1	Day 2	Day 3	...	Day 30
CL2	0	0	0	...	0
EDC	1288,290	1030,632	1108,013	...	1394,640
HCL	474,975	379,980	408,51	...	514,185
VCM	813,314	650,651	699,503	...	880,455
ETHYL	0	0	0	...	0

TAB. 8.5 – Quantités produites

Le dernier tableau 8.6 fournit une projection du niveau des stocks au cours du mois.

Stock level	Day 1	Day 2	Day 3	...	Day 30
CL2	0	0	0	...	0
EDC	3000	3000	3000	...	3000
HCL	0	0	0	...	0
VCM	2578,571	2334,051	2136,847	...	2000
ETHYL	0	0	0	...	0

TAB. 8.6 – Niveau du stock

Plusieurs conclusions peuvent être tirées à partir de ces deux derniers tableaux.
Premièrement, le niveau de stock de VC et d'EDC tendent vers le stock de sécurité. Ceci est évidemment attendu et souhaité puisque le stock de sécurité représente par définition la situation idéale. Les stocks de CL2 et d'HCL sont nuls ce qui correspond aux contraintes d'interdiction de stockage de ces substances.
Cependant, en début de mois, on constate que le niveau de stock de VC est supérieur au stock de sécurité. Ceci est dû au fait qu'en fin de mois, le niveau de production est saturé pour le VCM comme l'indique le tableau 8.2. Puisque la satisfaction de la demande est prioritaire sur le maintien d'un stock inférieur au stock de sécurité, contrainte elle-même prioritaire sur le maintien d'un stock plus élevé que le stock de sécurité, la solution retenue par l'optimiseur est donc celle d'une surproduction en début de mois permettant de satisfaire la demande durant tout le mois sans jamais devoir descendre en-dessous du stock de sécurité.
On constate par contre que le production d'EDC peut se faire réellement à flux tendu puisque le niveau d'utilisation des ressources n'est pas saturé. La

production varie donc de manière parallèle à la demande tout en maintenant le stock au niveau optimal, c'est-à-dire égal au stock de sécurité.
Une information fournie ici est donc qu'il probable que le niveau de production du VC soit insuffisant.

Chapitre 9

Autre exemple : Une soudière

9.1 Introduction

Le carbonate de sodium (Na_2CO_3) est un produit synthétique obtenu par la combinaison de deux composants : le calcaire (ou carbonate de calcium) et le sel (ou chlorure de sodium). Dans le procédé Solvay de production du carbonate de sodium, le carbonate est issu du calcaire et le sodium est issu du sel suivant la réaction chimique théorique suivante :

$$CaCO_3 + 2NaCl \Rightarrow Na_2CO_3 + CaCl_2 \tag{9.1}$$

Cette réaction se fait en pratique en plusieurs étapes et nécessite l'utilisation d'un intermédiaire chimique, l'ammoniac (NH_3) qui intervient en circuit fermé dans le cycle de fabrication.

Au terme de la réaction est produit, en plus du carbonate de sodium, un coproduit composé des deux autres éléments contenu dans le calcaire et le sel, le chlorure de calcium. Celui-ci, faute de valorisation possible, est destiné à être rejeté après traitement.

Le produit fini se présente sous deux formes distinctes :
- *Le carbonate de sodium léger* qui est largement utilisé comme matière première dans l'industrie pour ses multiples propriétés en tant qu'agent alcalin, fondant chimique, régulateur de pH, source de CO_2, ...
- *Le carbonate de sodium dense* (à granulométrie plus importante) qui est adapté aux besoins de l'industrie verrière en tant que fondant chimique de la silice (sable). Il trouve également de nombreuses applications dans les industries de la chimie, de la métallurgie et de la détergence.

A partir du carbonate de sodium léger, Solvay fabrique également du bicarbonate de sodium ($NaHCO_3$). Les débouchés du bicarbonate sont mul-

tiples :

- Industrie pharmaceutique (perfusion, comprimés effervescents, dialyse, ...) ;
- Alimentation animale (complément nutritionnel pour ruminants, porcs, volailles, ...) ;
- Alimentation humaine (levures, ...) ;
- Applications industrielles (industrie chimique, traitement de l'eau, détergence, ...) ;
- Procédé NEUTREC®(épuration des fumées dans l'incinération des déchets ménagers, hospitaliers et industriels, industrie métallurgique, fabrication de céramique, ...)

9.2 Fabrication

Nous analyserons ici le processus de fabrication à l'usine de Rheinberg en Allemagne.

La saumure extraite des sondages de la mine salière de Borth est traitée dans des bassins d'épuration afin d'éliminer les "impuretés" qui sont séparées par décantation. Cette saumure vierge est mélangée avec de la soude légère pour donner de la saumure épurée : c'est la **phase d'épuration**.
Celle-ci est mise en contact dans des colonnes de carbonatation avec le gaz carbonique libéré par le calcaire lors de sa cuisson dans les fours à chaux : c'est la **phase de carbonatation** [1] [2]. La réaction chimique entraîne la formation d'un brouet (mélange de liquide et de solide) contenant du bicarbonate de sodium.
Ce brouet est filtré afin de séparer le liquide du solide : c'est la **phase de filtration**.

La partie liquide (eaux-mères) qui contient l'ammoniac est traitée à l'aide de lait de chaux dans les distilleurs. L'ammoniac qui se dégage est récupéré dans sa totalité et repart dans le cycle de fabrication [3]. Les résidus de distillation (encore appelés "liquides rejets" constitués notamment de chlorure de calcium et de chlorure de sodium) sont envoyés en Saline dans des détendeurs pour y récupérer de la vapeur d'eau.

La partie solide, le bicarbonate de sodium brut, est calcinée dans des sécheurs : c'est **la phase de calcination**[4]. Sous l'action de la chaleur, le bicarbonate donne le carbonate de sodium léger. Après un nouveau traitement

[1] $CaCO_3 \rightarrow CaO + CO_2$
[2] $NaCl + CO_2 + NH_3 + H_2O \rightarrow NaHCO_3 + NH_4Cl$
[3] $2NH_4Cl + CaO \rightarrow 2NH_3 + CaCl_2 + H_2O$
[4] $2NaHCO_3 \rightarrow Na_2CO_3 + CO_2$

(hydratation puis calcination), la majeure partie de celui-ci est transformée en carbonate de sodium dense (à granulométrie plus importante) : c'est la **phase de densification**.

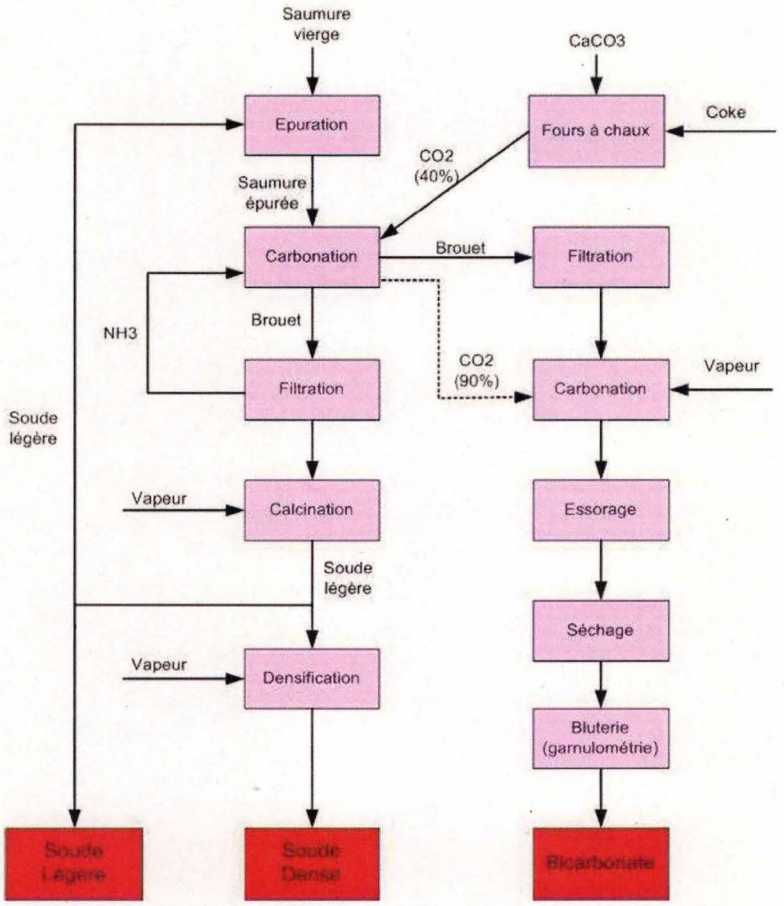

FIG. 9.1 – Fonctionnement d'une soudière

Une partie du bicarbonate de sodium brut est envoyé dans l'installation de "Bicarbonate Raffiné". Il est alors calciné ensuite filtré afin de garantir sa qualité et enfin envoyé dans des colonnes de carbonatation en inox où il est mit en contact avec du gaz carbonique[5]. Il se forme alors un brouet de bicarbonate de sodium (mélange de cristaux de bicarbonate et d'eaux mères). Après avoir été soutiré de la colonne, celui-ci est dirigé vers un bac épaississeur alimentant des essoreuses centrifuges où les eaux mères sont séparées du solide. Après essorage, le bicarbonate, encore légèrement humide, est en-

[5] $Na_2CO_3 + CO_2 + H_2O \rightarrow 2NaHCO_3$

voyé dans un sécheur où il est mis en contact avec de l'air chaud. (Rant, 1968)

9.3 Modélisation

La modélisation du problème sera ici simplifiée pour les raisons suivantes :

- l'objectif ici n'étant pas d'avoir un plan d'approvisionnement en matière première, celles-ci ne seront pas prises en compte dans le modèle. Ainsi, le four à chaux qui produit le CO2 ne sera pas modélisé. On considère en effet que les matières nécessaires à sa production seront toujours en quantité suffisante. De plus, son coût de production est négligeable.
- même si c'est le brouet et non la soude légère qui est la matière première du bicarbonate (cfr supra), nous travaillerons en équivalent soude légère ;

Dans notre modèle, l'ensemble des infrastructures nécessaires à la production de bicarbonate, l'ensemble des infrastructures nécessaires à la production de soude légère et l'ensemble des infrastructures nécessaires à la production de soude dense seront regroupées en trois ressources distinctes. Par contre, l'épuration de la saumure sera modélisée comme une ressource séparée puisqu'elle est vendue et qu'elle consomme de la soude légère.

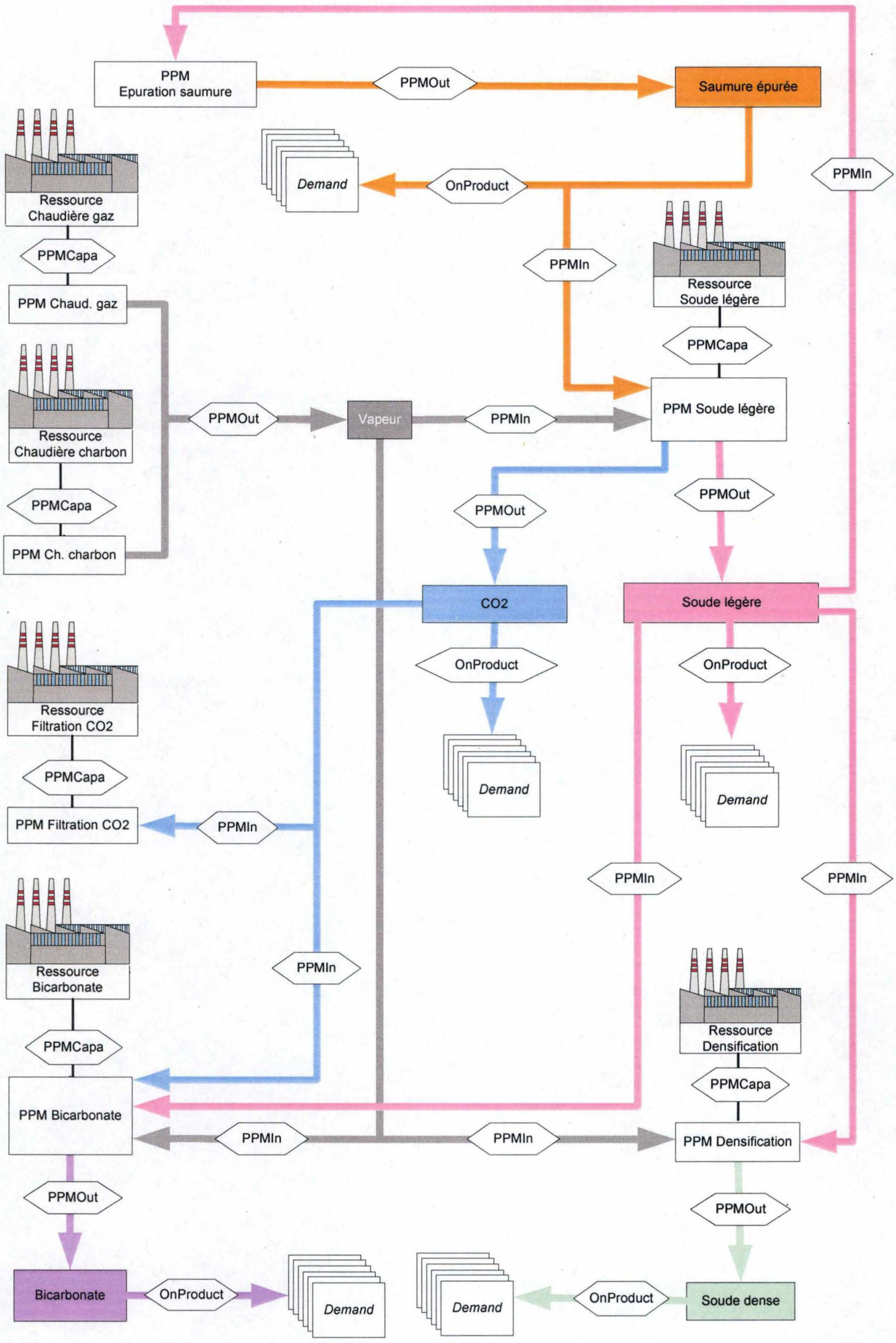

FIG. 9.2 - Modélisation de la production de soude

Comme pour la modélisation de la production de VC, chaque produit intervenant dans la production de soude est symbolisé par un cadre. Cependant, dans un souci de clarté, nous n'avons plus mentionné ici les différentes caractéristiques propres à chaque produit. Le lecteur intéressé trouvera en annexe ces informations.

Dans la figure 9.2, nous voyons que le bicarbonate, la soude dense, la soude légère, le C02 et la saumure épurée sont vendus. Pour produire du bicarbonate, nous avons besoin de vapeur, de soude légère et de CO2. Pour produire de la soude dense, nous avons besoin de soude légère et de vapeur. Lors de la fabrication de la soude légère, de la saumure épurée est utilisée et le CO2 est fabriqué comme co-produit. La production de vapeur se fait soit sur une chaudière au charbon, soit sur une chaudière au gaz. Enfin, nous voyons que pour produire de la saumre épurée, nous avons besoin de soude légère. Pour un souci de clarté du schéma, les valeurs et caractéristiques des PPMs n'ont pas été reprises. Le lecteur intéressé trouvera en annexe ces informations.

Comme pour le VC, toute nos ressources n'auront qu'un seul PPM. Les carcatéristiques de chaque ressource seront reprises en annexe.

A nouveau, l'ensemble de ces données sont entrées dans l'optimiseur sous forme de deux fichier XML.

9.4 Résultats

L'utilisateur devra spécifier à l'optimiseur les différentes contraintes qui s'appliquent sur les produits et les ressources sous forme de fichier XML. Sur base de ces données (reprises en annexes), l'optimiseur va fournir un plan de production et un projection de la production.

Plan de production

Le tableau 9.1 fournit la quantité de matière première à acheter. Vu que, dans l'exemple qui nous occupe, les matières premières ne sont pas gérées par l'optimiseur, celles-ci ne sont pas reprises dans le tableau. De plus, les autres produits nécessaires à la production de soude sont directement fabriqués par Solvay et ne sont pas achetés.

Le tableau 9.2 donne les indications quant au niveau d'utilisation à appliquer à chaque ressource. Ces données sont exprimées en nombre d'heures par jour.

Procurement	Day 1	Day 2	Day 3	Day 4	...	Day 30
SE	0	0	0	0	...	0
VE	0	0	0	0	...	0
SL	0	0	0	0	...	0
SD	0	0	0	0	...	0
BIR	0	0	0	0	...	0
C02	0	0	0	0	...	0

TAB. 9.1 – Quantité à acheter

Ressource	R. Lev D. 1	R. Lev D. 2	R. Lev D. 3	R. Inc D. 4	R. Inc D. 30
RES_S2_SE	2.017	2.017	1.98	...	1.92
RES_VE_CHARB	24	24	24	...	24
RES_VE_GAZ	1.32	1.32	1.15	...	0.85
RES_SL	20.92	20.92	20.46	...	19.61
RES_SD	23.23	23.23	22.65	...	21.6
RES_BIR	24	24	23	...	21.16

TAB. 9.2 – Niveau d'utilisation des ressources

Le tableau 9.2 nous montre qu'il est impossible de produire du bicarbonate de manière maximale en fin de période car sa production nécessite du CO2. Or pour disposer de C02, il faut produire de la soude. Or, les niveaux de stock de soude sont au maximum.
On constate également qu'une incapacité de production de bicarbonate d'une heure au troisème jour a un impact sur toute la chaine.

Projections établies

Les tableaux 9.3 et 9.4 nous permettent de constater que l'optimisation proposée ici sur base des contraintes spécifiées par l'utilisateur parvient à satisfaire la totalité de la demande.

Le tableau 9.5 donne une prévision quant à la quantité fabriquée pour chaque produit.

Le tableau 9.6 fournit une projection des niveaux de stock. Il nous permet de constater que les stocks de soude légère et de soude dense arrivent rapidement à saturation. Ceci est dû au fait que la demande en soude légère et en soude dense est faible mais que la production de bicarbonate nécessite du CO2 qui n'est produit que lorsque l'on fabrique de la soude légère. Le stock de soude légère augmente donc sans être écoulé. De plus, cette soude légère est densifiée en soude dense pour être stockée. Cette soude dense est également peu écoulée.

Satisfied Demand	Day 1	Day 2	Day 3	Day 4	...	Day 30
SE	0	0	0	0	...	0
VE	0	0	0	0	...	0
SL	0	0	0	0	...	0
SD	234,743	895,171	896,708	896,708	...	880,455
BIR	0	0	0	0	...	0
C02	0	0	0	0	...	0

TAB. 9.3 – Demandes satisfaites

Unsatisfied Demand	Day 1	Day 2	Day 3	Day 4	...	Day 30
SE	2000	2000	2000	2000	...	2000
VE	0	0	0	0	...	0
SL	200	200	200	200	...	200
SD	900	900	900	900	...	900
BIR	160	160	160	160	...	160
C02	80	80	80	80	...	80

TAB. 9.4 – Demandes non satisfaites

Code	Prod D. 1	Prod D. 2	Prod D. 3	...	Prod D. 30
SE	8407.69	8407.69	8266.34	...	8007.58
VE	3269.23	3269.23	3197.11	...	3065.09
SL	1307.69	1307.69	1278.84	...	1226.03
SD	968.15	968.15	944.07	...	900.00
BIR	150	150	143.75	...	132.30
CO2	170	170	166.25	...	159.38

TAB. 9.5 – Niveau de production

Code	Stock D. 1	Stock D. 2	Stock D. 3	...	Stock D. 30
SE	0	0	0	...	0
VE	0	0	0	...	0
SL	4000	4000	4000	...	4300
SD	4068.15	4136.30	4180.38	...	4400
BIR	990	980	963.75	...	350.95
CO2	0	0	0	...	0

TAB. 9.6 – Niveau de stock

Chapitre 10

Evaluation de l'outil

Dans ce chapitre, nous voudrions mentionner les avantages et les inconvénients les plus marquants de notre outil. Il nous semble en effet important de considérer ce travail comme le début d'une réflexion sur la modélisation des processus de production plutôt que comme un aboutissement en soi. Plusieurs améliorations pourraient en effet être proposées mais nécessiteraient une connaissance plus précise des moyens de productions actuels et sortiraient du cadre de ce travail. Il nous a paru cependant intéressant d'en faire mention dans ce chapitre.

10.1 Avantages de l'outil

Le principal avantage de notre outil est sa capacité d'adaptation à différents scenarii de production. Nous l'avons démontré ici dans deux situations, mais l'outil est a priori conçu de telle manière qu'il soit possible de l'étendre à bien d'autres plans de production. Cependant, seule une application à l'ensemble des plans de production d'un groupe industriel tel que Solvay permettrait de démontrer de manière certaine une totale adaptabilité. Il est bien évident que cet exercice sortait totalement du cadre de ce travail.

La réactivité par rapport aux évenements nous semble également un élément crucial dans l'utilisation du modéliseur au sein d'une chaine de production. En effet, tout évenement intercurrent modifiant les paramètres de production tel que, par exemple, une défectuosité d'une ressource, peut être facilement intégré dans le modéliseur qui va alors modifier en fonction la solution proposée.

Un autre intérêt de l'outil est qu'il fournit non seulement les données nécessaires pour appliquer le plan de production mais aussi une prévision quant aux résultats de la production. Ceci permet à l'utilisateur de s'assurer que les différentes contraintes seront respectées dans la mesure des possibilités.

La mise en place dans le paysage informatique est aisée puisque l'interface utilisée est tout à fait standard et répandue.

L'utilisation est aisée et ne nécessite aucun apprentissage particulier. Quant à la présentation actuelle des résultats, elle est simple et conviviale.

10.2 Inconvénients de l'outil et améliorations possibles

L'impossibilité d'adaptation à une solution suboptimale constitue certainement la limitation majeure de notre outil. En effet, sur base des contraintes imposées par l'utilisateur, le modéliseur fournit une seule solution qui correspond aux critères fixés dans les équations et qui aboutissent à une production optimale. Si, pour l'une ou l'autre raison, l'utilisateur ne souhaite pas avoir un plan de production tout à fait optimal, le modéliseur ne lui permet pas cette option.

Un des incovénients de notre outil est que les prévisions qu'il fournit, se base sur les mêmes équations que celles qui sont utilisées pour déterminer les besoins de production. Il n'y a donc pas de système indépendant permettant de vérifier l'exactitude du plan de production quant aux résultats attendus.

L'utilisation de variables en nombres entiers est problématique avec l'algorithme du Simplex. En effet, nous utilisons alors la stratégie Branch & Bound qui augmente considérablement les temps de calcul de la solution. Pour exemple, en rajoutant trois contraintes qui utilisent des variables en nombres entiers, le temps d'éxécution fût d'environ 4 heures contre environ 10 secondes sans ces contraintes. Il serait certainement intéressant de rajouter une limitation dans le temps de calcul qui se baserait soit sur l'obtention d'une solution fixée à un certain pourcentage de la solution optimale, soit sur un temps absolu de calcul (par exemple 5 minutes).

Enfin, aucun ordonnancement ne peut être géré par le modéliseur. Il est impossible à l'utilisateur de fixer un ordre dans la production des différents produits finis sur une même ressource, cette exigence pouvant cependant revêtir une importance cruciale. En effet, il peut être plus efficace de fabriquer un produit avant un autre par exemple pour des raisons de maintenance (nettoyage de l'outil, ...).

Chapitre 11

Conclusions

En conclusion, notre étude confirme que l'optimisation de la planification de production en programmation linéaire est réalisable. Les équations développées semblent être utilisables et généralisables aux différents modèles de production étudiés.

Le module présenté ici se veut potentiellement applicable à tous types de production continue et permet de calculer un plan de production. Ce plan permettant au responsable de production d'adapter différents paramètres afin de réduire les coûts finaux. L'objectif du travail a donc été atteint.

Il nous aurait paru particulièrement instructif de pouvoir comparer les coûts de production engendrés par une solution proposée par notre modéliseur avec les coûts réels actuels. Nous aurions par exemple pu comparer sur une année les différents coûts mensuels et identifier une éventuelle différence statistique entre ces coûts. Malheureusement, les coûts actuels de production sont indisponibles pour des raisons évidentes de confidentialité.

Bien sûr, l'efficacité de cet outil ne pourra être réellement démontrée que par son application en grandeur réelle, au sein même des processus de fabrication, soumis aux différentes contraintes et aléas de la production industrielle.

Bibiliographie

1. Guéret Ch., Prins Ch., Sevaux M., *Programmation linéaire.* Editions Eyrolles, 2000, ISBN 2-212-09202-4

2. Knolmayer G., Mertens P., Zeier A., *Supply Chain Management Based on SAP Systems.* Springer, 2001, ISBN 3-540-66952-3

3. Teghem J., *Programmation linéaire.* Editions Ellipses, 1996, ISBN 2-8004-1317-4

4. Dantzig G.B, *Problems for the Numerical Analysis of the Future.* Proceedings of the Symposium on Modern Calculating Machines and Numerical Methods, UCLA 1948

5. Dantzig George B., Thapa Mukund N., *Linear Programming 1. Introduction.* Springer series in operations research, 1997, ISBN 0-387-94833-3

6. Dantzig George .B, Thapa Mukund N., *Linear Programming 2. Theory and Extensions.* Springer series in operations research, 2003, ISBN 0-387-98613-8

7. Wolsey L., *Integer Programming.* Wiley-Interscience, 1998, ISBN 0-4712-8366-5

8. Faure Robert, Lemaire Bernard, Picouleau Christophe, *Précis de recherche opérationnelle.* Dunod, 2000, ISBN 2-10-004691-8

9. Lougee-Heimer Robin, *The Common Optimization INterface for Operations Research.* IBM Journal of Research and Development,vol. 47(1) :57-66, January 2003. (Accessed March 2005 at http ://www.research.ibm.com/journal/rd/471/lougee.html)

10. Fourer R., Gay David M., Kernighan Brian W. *AMPL : A Modeling Language for Mathematical Programming*, Brooks/Cole Publishing Company, 2002, ISBN 0-534-38809-4

11. Verbois S. *Programmation linéaire.* FUNDP 2004

12. L. Ladanyi, *Parallel Branch and Cut and Its Application to the Traveling Salesman Problem*, PhD thesis, Cornell University, May 1996.

13. Makhorin Andrew, *GNU Linear Programming Kit*, Draft Edition, 2005 (Accessed March 2005 at http ://www.gnu.org/software/glpk/glpk.html)

14. Fourer R. *Linear Programming Eighth in a series of LP surveys highlights recent trends in profession's most popular software.* OR/MS Today June 2005 (Accessed June 2005 at http ://www.lionhrtpub.com/orms/surveys/LP/LP-survey.html)

15. Linderoth J. T., Ralphs T.K. *Noncommercial Software for Mixed-Integer Linear Programming.* Lehigh University, 2004 (Accessed July 2005 at http ://www.lehigh.edu/ tkr2/research/papers/MILP04.pdf)

16. Cooper Clark, *A performance comparison of six stream-oriented XML parsers.* 1999 (Accessed May 2005 at http ://www.xml.com/pub/a/Benchmark/article.html)

17. Samii Alexandre., *Stratégies logistiques. Fondements, méthodes et applications.* Editions DUNOD, 2001, ISBN 2-10-005557-7

18. Gaspard P., *Gestion des stocks et de la production.* ULB 1997

19. Eymery P., *La logistique de l'entreprise. Supply chain management.* Editions Hermes, 1997, ISBN 2-86601-589-4

20. Rant Z., *Die Erzeugung von Soda nach dem Solvay-Verfahren.* 19 Seiten - Enke, 1968, ISBN B0000BT52B

Annexes

Le fichier des données maîtres

On appelle données maîtres les données qui varient peu dans le temps. Elles regroupent les contraintes liées intrinsèquement aux types de production, aux produits et aux équipements utilisés dans la production. Ce sont ces données qui permettent de modéliser une production. Nous avons ici repris le fichier XML des données maîtres qui décrit la production de VC.

```
<?xml version="1.0"?>
<master_data nom="VC">
  <!-- "Les produits" -->
  <produit id="CL2">
    <ssl>0</ssl>
    <stockmax>0</stockmax>
    <storcost>0</storcost>
    <ssc>0</ssc>
    <nondelpen>3000</nondelpen>
    <proccost>0</proccost>
    <procsnppen>0</procsnppen>
        <minprocpen>0</minprocpen>
    <procbatchsize>1</procbatchsize>
    <procbatchpen>1</procbatchpen>
  </produit>
  <produit id="EDC">
    <ssl>3000</ssl>
    <stockmax>7000</stockmax>
    <storcost>0.043</storcost>
    <ssc>200</ssc>
    <nondelpen>3000</nondelpen>
    <proccost>9999</proccost>
    <procsnppen>0</procsnppen>
    <minprocpen>0</minprocpen>
    <procbatchsize>1</procbatchsize>
    <procbatchpen>1</procbatchpen>
  </produit>
  <produit id="HCL">
    <ssl>0</ssl>
```

```
    <stockmax>0</stockmax>
    <storcost>0</storcost>
    <ssc>0</ssc>
    <nondelpen>3000</nondelpen>
    <proccost>9999</proccost>
    <procsnppen>0</procsnppen>
    <minprocpen>0</minprocpen>
    <procbatchsize>1</procbatchsize>
    <procbatchpen>1</procbatchpen>
  </produit>
  <produit id="VCM">
    <ssl>2000</ssl>
    <stockmax>4000</stockmax>
    <storcost>0.080</storcost>
    <ssc>250</ssc>
    <nondelpen>4000</nondelpen>
    <proccost>9999</proccost>
    <procsnppen>0</procsnppen>
    <minprocpen>0</minprocpen>
    <procbatchsize>1</procbatchsize>
    <procbatchpen>1</procbatchpen>
  </produit>
  <produit id="ETHYL">
    <ssl>0</ssl>
    <stockmax>0</stockmax>
    <storcost>0</storcost>
    <ssc>0</ssc>
    <nondelpen>3000</nondelpen>
    <proccost>0</proccost>
    <procsnppen>0</procsnppen>
    <minprocpen>0</minprocpen>
    <procbatchsize>1</procbatchsize>
    <procbatchpen>1</procbatchpen>
  </produit>

  <!-- "Les ressources" -->
  <ressource id="EDC_CHLO">
    <maxrescap>24</maxrescap>
    <maxvar>20</maxvar>
    <minrescap>12</minrescap>
    <minrespen>100</minrespen>
  </ressource>
  <!-- Les ressources -->
  <ressource id="EDC_OXY">
    <maxrescap>24</maxrescap>
    <maxvar>20</maxvar>
    <minrescap>12</minrescap>
    <minrespen>100</minrespen>
  </ressource>
```

```
  <!-- Les ressources -->
  <ressource id="VCM">
    <maxrescap>24</maxrescap>
    <maxvar>20</maxvar>
    <minrescap>12</minrescap>
    <minrespen>100</minrespen>
  </ressource>

  <!-- "Les ppms" -->
  <ppm nom="V_CHLO">
    <cost>10</cost>
    <ppmin id="CL2">0.705</ppmin>
    <ppmin id="ETHYL">0.292</ppmin>
    <ppmout prod="EDC">1</ppmout>
    <ppmcapa ress="EDC_CHLO">92.73</ppmcapa>
  </ppm>

  <ppm nom="V_OXY">
    <cost>20</cost>
    <ppmin id="HCL">0.774</ppmin>
    <ppmin id="ETHYL">0.303</ppmin>
    <ppmout prod="EDC">1</ppmout>
    <ppmcapa ress="EDC_OXY">75.09</ppmcapa>
  </ppm>

  <ppm nom="V_VCM">
    <cost>26.9</cost>
    <ppmin id="EDC">1.584</ppmin>
    <ppmout id="HCL">0.584</ppmout>
    <ppmout prod="VCM">1</ppmout>
    <ppmcapa ress="VCM">95.57</ppmcapa>
  </ppm>
</master_data>
```

Le fichier des données transactionnelles

Les données transactionnelles sont les données qui varient d'un plan de production à l'autre. Le fichier XML des données transactionnelles doit être fourni par l'utilisateur afin que le module puisse calculer un plan de production optimum. Nous avons repris ici les données transactionnelles pour une production de VC.

```
<?xml version="1.0"?>
<transactional_data nom="VC">
  <!-- "Les produits" -->
  <produit id="CL2">
    <stockinit>0</stockinit> <!--Le stock initial -->
    <procsnp>0</procsnp>   <!--La quantite d achats obligatoires -->
    <minproc>0</minproc>   <!--La quantite minimum aäacheter -->
  </produit>
  <produit id="EDC">
    <stockinit>3000</stockinit> <!--Le stock initial -->
    <procsnp>0</procsnp>   <!--La quantite d achats obligatoires -->
    <minproc>0</minproc>   <!--La qunatite minimum aäacheter -->
  </produit>
  <produit id="HCL">
    <stockinit>0</stockinit> <!--Le stock initial -->
    <procsnp>0</procsnp>   <!--La quantite d achats obligatoires -->
    <minproc>0</minproc>   <!--La qunatite minimum aäacheter -->
  </produit>
  <produit id="VCM">
    <stockinit>2000</stockinit> <!--Le stock initial -->
    <procsnp>0</procsnp>   <!--La quantite d achats obligatoires -->
    <minproc>0</minproc>   <!--La quantite minimum aäacheter -->
    <dem j="1">234.743</dem> <!--La demande au jour 1 -->
    <dem j="2">895.171</dem> <!--La demande au jour 2 -->
    <dem j="3">896.708</dem> <!--La demande au jour 3 -->
    <dem j="4">896.708</dem> <!--La demande au jour 4 -->
    <dem j="5">896.708</dem> <!--La demande au jour 5 -->
    <dem j="6">896.708</dem> <!--La demande au jour 6 -->
    <dem j="7">896.708</dem> <!--La demande au jour 7 -->
    <dem j="8">896.708</dem> <!--La demande au jour 8 -->
    ...
    <dem j="28">880.455</dem> <!--La demande au jour 18 -->
    <dem j="29">880.455</dem> <!--La demande au jour 19 -->
    <dem j="30">880.455</dem> <!--La demande au jour 20 -->
  </produit>
  <produit id="ETHYL">
    <stockinit>0</stockinit> <!--Le stock initial -->
    <procsnp>0</procsnp>   <!--La quantite d achats obligatoires -->
    <minproc>0</minproc>   <!--La quantite minimum aäacheter -->
  </produit>
```

```
41
42    <!-- "Les ressources" -->
43    <ressource id="EDC_CHLO">
44      <reslevin>20</reslevin>
45      <resinc j="1">1</resinc> <!--Le nbre d heures d incapacite au jour 1 -->
46      <resinc j="6">2</resinc> <!--Le nbre d heures d incapacite au jour 6 -->
47      <resinc j="8">1</resinc> <!--Le nbre d heures d incapacite au jour 8 -->
48      <resinc j="15">1</resinc> <!--Le nbre d heures d inc. au jour 15 -->
49      <resinc j="21">1</resinc> <!--Le nbre d heures d inc. au jour 21 -->
50      <resinc j="28">1</resinc> <!--Le nbre d heures d inc. au jour 28 -->
51    </ressource>
52    <ressource id="EDC_OXY">
53      <reslevin>16</reslevin>
54      <resinc j="2">1</resinc> <!--Le nbre d hrs d incapacites au jour 2 -->
55      <resinc j="26">1</resinc> <!--Le nbre d hrs d incapacites au jour 26 -->
56    </ressource>
57    <ressource id="VCM">
58      <reslevin>24</reslevin>
59      <resinc j="3">1</resinc> <!--Le nb d heures d incapacites au jour 3 -->
60      <resinc j="21">1</resinc> <!--Le nb d h d incapacites au jour 21 -->
61      <resinc j="23">1</resinc> <!--Le nb d h d incapacites au jour 23 -->
62    </ressource>
63
64  </transactional_data>
```

Fichier modèle

Le fichier modèle est le fichier qui contient les équations et qui est passé au solveur GLPK. Les équations présentes dans ce fichier dépendent du fichier XML de paramétrisation.

```
#Equations de mon mémoire cfr fichier pdf

/*************************************************************************/
/* Déclaration des données */
/*************************************************************************/

# Jour      = j allant de 1..M
# Produits  = a  allant de 1..N
# Ressources = r allant de 1..P
# PPMs      = b allant de 1..O


set N; # Ensemble des produits
set P; # Ensemble des ressources
set O; # Ensemble des PPMs

#param m := 5; # Nombre de jours dans l horizon de planification
param m integer;

param M := 9999; # Variable mathématique devant être suffisamment grande
param G := 9999; # Variable mathématique devant être suffisamment grande

/* Les données : */
# Les données relatives au produit :
/* Niveau du Safety Stock pour chacun des produits (en Tonne)*/
param SafetyStockLevel{a in N};

/* Niveau du Stock Maximum pour chacun des produits (en Tonne)*/
param StockMaxi{a in N};

/* Couts pas unité de stockage */
param StorageCost{a in N};

/* Couts en cas de franchissement du stock de sécurité */
param SafetyStockCost{a in N};

/* Couts en cas de demande non satisfaite */
param NonDelPen{a in N};

/* Couts d'achat        */
param ProcCost{a in N};

/* Le stock initial */
param StockInitial{a in N};
```

```
45
46 /* La quantitée d'achat prévue par SNP */
47 param ProcSNP{a in N};
48
49 /* La pénalité en cas de non respect de la quantitée à acheter prévue par SNP*/
50 param ProcSNPPen{a in N};
51
52 /* La quantitée minimum a acheter */
53 param MinProc{a in N};
54
55 /* La pénalité en cas de non respect de la quantitée minimum à acheter*/
56 param MinProcPen{a in N};
57
58 /* La taille du lot à acheter */
59 param ProcBatchSize{a in N};
60
61 /* La pénalité en cas de non respect de la taille du lot à acheter */
62 param ProcBatchPen{a in N};
63
64 # Les données relatives aux ressources
65 /* Niveau initial de production sur les ressources */
66 param ResLevIn{r in P};
67
68 /* Capacités maximales des ressources (en heure) */
69 param MaxResCap{r in P};
70
71 /* Variation Maximale pour une ressource (en %) */
72 param MaxVar{r in P};
73
74 /* Les périodes d'arrêts des ressources (en heure) */
75 param ResInc{r in P, j in 1..m};
76
77 /* le minimum de production */
78 param MinResCap{r in P};
79
80 /* Pénalité associé au non respect du minimum de production */
81 param MinResPen{r in P};
82
83 # Les données relatives aux PPMs
84 /* Cout d'utilisation du PPM */
85 param PPMCost{b in O};
86
87 /* La quantitée de l'article n Consommé par la ppm o */
88 param PPMIn{b in O, a in N};
89
90 /* La quantitée de l'article n produite par la ppm o */
91 param PPMOut{b in O, a in N};
92
93 /* Capacité d'un PPM (secondes) sur la ressource r */
```

```
94  param PPMCapa{b in O, r in P};
95
96  # Les demandes
97  /* La demande du produit a en période j */
98  param Demande{a in N,j in 1..m};
99
100
101 /*************************************************************************/
102 /* Les variables */
103 /*************************************************************************/
104
105 /* Quantitée stockée du produit a au jour j */
106 var stocklevel{a in N,j in 1..m}, >= 0;
107
108 /* Demande satisfaite du produit a au jour j */
109 var satdem{a in N,j in 1..m}, >= 0;
110
111 /* Demande non-satisfaite du produit a au jour j */
112 var unsatdem{a in N,j in 1..m}, >= 0;
113
114 /* Demande dependante sur le produit a au jour j */
115 var depdem{a in N,j in 1..m}, >= 0;
116
117 /* Quantitée achetée du produit a au jour j */
118 var proclevel{a in N,j in 1..m}, >= 0;
119
120 /* Taux d'utilisation du PPM p au jour j */
121 var ppmlevel{b in O,j in 1..m}, >= 0;
122
123 /* Représente la quantitée produite du produit a au jour j */
124 var prodlevel{a in N,j in 1..m}, >= 0;
125
126 /* Niveau de production sur une ressource r au jour j */
127 var reslevel{r in P,j in 1..m}, >= 0;
128
129 /* Stock de sécurité virtuel négatif du produit a au jour j */
130 var alpha{a in N,j in 1..m}, >= 0;
131
132 /* Stock de sécurité virtuel positif du produit a au jour j */
133 var beta{a in N,j in 1..m}, >= 0;
134
135 /* Différence positive de quantitée achetée par rapport aux prévisions de SNP*/
136 var gamma{a in N}, >= 0;
137
138 /* Différence négative de quantitée achetée par rapport aux prévisions de SNP*/
139 var delta{a in N}, >= 0;
140
141 /* Différence avec la quantitée minimum à acheter */
142 var minproclevel{a in N,j in 1..m}, >= 0;
```

```
143
144 /* Variable binaire pour l'achat minimum */
145 var x{a in N,j in 1..m}, binary;
146
147 /* Quantitée manquate pour arriver à un lot complet */
148 var batchpenality{a in N,j in 1..m}, >= 0;
149
150 /* Le nombre de lots à acheter le jour j */
151 var numberofbatches{a in N,j in 1..m}, integer;
152
153 /* Capacité manquante pour atteindre la capacité de production minimum */
154 var minreslevel{r in P,j in 1..m}, >= 0;
155
156 /* Variable binaire pour la production minimum */
157 var y{r in P,j in 1..m}, binary;
158
159 /*************************************************************************/
160 /* La fonction objectif */
161 /*************************************************************************/
162
163 # Minimiser la somme des couts de production et de stockage
164 /* Couts de production*/
165 minimize coutTotal: sum{b in 0,j in 1..m} ppmlevel[b,j] * PPMCost[b]
166
167 /* Couts de stockage */
168           + sum{a in N,j in 1..m} stocklevel[a,j] * StorageCost[a]
169
170 /* Couts en cas de franchissement du Safety Stock */
171           + sum{a in N,j in 1..m} beta[a,j] * SafetyStockCost[a]
172
173 /* Couts en cas de demande non satisfaite */
174           + sum{a in N,j in 1..m} unsatdem[a,j] * NonDelPen[a]
175
176 /* Couts d'approvisionnement */
177           + sum{a in N,j in 1..m} proclevel[a,j] * ProcCost[a]
178
179 /* Couts en cas de non respect des quantités à acheter prévue par SNP */
180           + sum{a in N} delta[a] * ProcSNPPen[a]
181
182 /* Couts en cas de non respect de la quantité minimum à acheter*/
183           + sum{a in N,j in 1..m} minproclevel[a,j] * MinProcPen[a]
184
185 /* Couts en cas de non respect de la taille des lots à acheter*/
186           + sum{a in N,j in 1..m} batchpenality[a,j] * ProcBatchPen[a]
187
188 /* Couts en cas de fonctionnement d'une ressource en dessous du minimum */
189           + sum{r in P,j in 1..m} minreslevel[r,j] * MinResPen[r];
190
191 /*************************************************************************/
```

```
/* Les contraintes */
/*************************************************************************/
# 1) Contraintes d equilibre de stock. Premiere periode
s.t. EqStJ1{a in N}: StockInitial[a] + proclevel[a,1] + prodlevel[a,1] =
                  stocklevel[a,1] + satdem[a,1] + depdem[a,1];

# 1Bis) Autres periodes
s.t. EqStJM{a in N,j in 2..m}: stocklevel[a,j-1] + proclevel[a,j] + prodlevel[a,j] =
                           stocklevel[a,j] + satdem[a,j] + depdem[a,j];

# 2) Respect de la contrainte de stockage maximum
s.t. StockMax{a in N,j in 1..m}: stocklevel[a,j] <= StockMaxi[a];

# 3) Respect de la contrainte sur le Safety Stock
s.t. SafetyStock{a in N,j in 1..m}: stocklevel[a,j] - SafetyStockLevel[a] =
                                alpha[a,j] - beta [a,j];

# 4) Contrainte d equilibre de la demande
s.t. EqDem{a in N,j in 1..m}: Demande[a,j] = satdem[a,j] + unsatdem[a,j];

# 5) Respect des achats prévus par SNP
s.t. ProcSNPC{a in N}: sum{j in 1..m} proclevel[a,j] - ProcSNP[a] =
                                  gamma[a] - delta [a];

# 6) Contrainte de nomenclature : Production
s.t. NomProd{a in N,j in 1..m}: prodlevel[a,j] =
                           sum{b in O} ppmlevel[b,j] * PPMOut[b,a];

# 7) Contrainte de nomenclature : Consommation
s.t. NomCons{a in N,j in 1..m}: depdem[a,j] =
                           sum{b in O} ppmlevel[b,j] * PPMIn[b,a];

# 8) Determination du taux d utilisation d'une ressource r au jour j
s.t. ResDet{r in P,j in 1..m}: reslevel[r,j] = sum{b in O} ppmlevel[b,j] *
                                        ( PPMCapa[b,r] / 3600 );

# 9) Contrainte sur le maximum de Production
s.t. MaxRes{r in P,j in 1..m}: reslevel[r,j] <= MaxResCap[r] - ResInc[r,j];

# 10) Limitations des variations de production (vers le haut) Premiere période
s.t. VarResUPJ1{r in P}: reslevel[r,1] <=
                      ResLevIn[r] + ResLevIn[r] * MaxVar[r] / 100;

# 10 bis) Limitations des variations de production (haut) Autres périodes
s.t. VarResUPJM{r in P,j in 2..m}: reslevel[r,j] <=
                        reslevel[r,j-1] + reslevel[r,j-1] * MaxVar[r] / 100;

# 11) Limitations des variations de production (vers le bas) Premiere période
s.t. VarResDOWNJ1{r in P}: reslevel[r,1] >=
```

```
                        ResLevIn[r] - ResLevIn[r] * MaxVar[r] / 100;

# 11 bis) Limitations des variations de production (vers le bas) Autres périodes
s.t. VarResDOWNJM{r in P,j in 2..m}: reslevel[r,j] >=
                         reslevel[r,j-1] - reslevel[r,j-1] * MaxVar[r] / 100;

# 12) Taille des lots // CONTRAINTE NON OBLIGATOIRE
s.t. Batchsize{a in N,j in 1..m}: proclevel[a,j] =
                numberofbatches[a,j] * ProcBatchSize[a] - batchpenality[a,j];

# 13) Achats Minimums : Pas d achats // CONTRAINTE NON OBLIGATOIRE
s.t. MinProczero{a in N,j in 1..m}: proclevel[a,j] <= M * x[a,j];

# 13 Bis) Achats Minimum : il y a un achat // CONTRAINTE NON OBLIGATOIRE
s.t. MinProcmin{a in N,j in 1..m}: proclevel[a,j] >=
                             MinProc[a]+ M * (x[a,j]-1) - minproclevel[a,j];

# 14) Minimum de production : Pas de production // CONTRAINTE NON OBLIGATOIRE
s.t. MinReszeroC{r in P,j in 1..m}: reslevel[r,j] <= G * y[r,j];

# 14 Bis) Minimum de production : Production minimum // CONTRAINTE NON OBLIGATOIRE
s.t. MinResminC{r in P,j in 1..m}: reslevel[r,j] >=
                            MinResCap[r]+ G * (y[r,j]-1) - minreslevel[r,j];


end;
```

Fichier DATA

Le fichier *data* reprend l'ensemble des variables qui seront intégrées dans les équations contenues dans le fichier *modèle*. Nous reprenons ici le fichier *data* écrit par le module dans le cadre de l'optimisation de la production du VC.

```
/* Le fichier des données */
data;
# Les Ensembles
set N := CL2 EDC HCL VCM ETHYL ;
set P := EDC_CHLO EDC_OXY VCM ;
set O := V_CHLO V_OXY V_VCM ;
param m:=30;
# Les donnees relatives aux produits
param SafetyStockLevel:=
ETHYL 0.000000
VCM 2000.000000
HCL 0.000000
EDC 3000.000000
CL2 0.000000
;
param StockMaxi:=
ETHYL 0.000000
VCM 4000.000000
HCL 0.000000
EDC 7000.000000
CL2 0.000000
;
param StorageCost:=
ETHYL 0.000000
VCM 0.080000
HCL 0.000000
EDC 0.043000
CL2 0.000000
;
param SafetyStockCost:=
ETHYL 0.000000
VCM 250.000000
HCL 0.000000
EDC 200.000000
CL2 0.000000
;
param NonDelPen:=
ETHYL 3000.000000
VCM 4000.000000
HCL 3000.000000
EDC 3000.000000
CL2 3000.000000
```

```
;
param ProcCost:=
ETHYL 0.000000
VCM 9999.000000
HCL 9999.000000
EDC 9999.000000
CL2 0.000000
;
param StockInitial:=
ETHYL 0.000000
VCM 2000.000000
HCL 0.000000
EDC 3000.000000
CL2 0.000000
;
param ProcSNP:=
ETHYL 0.000000
VCM 0.000000
HCL 0.000000
EDC 0.000000
CL2 0.000000
;
param ProcSNPPen:=
ETHYL 0.000000
VCM 0.000000
HCL 0.000000
EDC 0.000000
CL2 0.000000
;
# Les donnees relatives aux ressources
param ResLevIn:=
VCM 24.000000
EDC_OXY 16.000000
EDC_CHLO 20.000000
;
param MaxResCap:=
VCM 24.000000
EDC_OXY 24.000000
EDC_CHLO 24.000000
;
param MaxVar:=
VCM 20.000000
EDC_OXY 20.000000
EDC_CHLO 20.000000
;
param ResInc:      1      2      3      4      5      6      7      8   ...
          EDC_CHLO 1.000000 0.000000 0.000000 0.000000 0.000000      ...
          EDC_OXY 0.000000 1.000000 0.000000 0.000000 0.000000       ...
          VCM 0.000000 0.000000 1.000000 0.000000 0.000000 0.000000 ...
```

```
92  ;
93  # Les donnees relatives aux ppms
94  param PPMCost:=
95  V_VCM 26.900000
96  V_OXY 20.000000
97  V_CHLO 10.000000
98  ;
99  param PPMIn:      CL2     EDC     HCL     VCM     ETHYL     :=
100           V_CHLO 0.705000 0.000000 0.000000 0.000000 0.292000
101           V_OXY 0.000000 0.000000 0.774000 0.000000 0.303000
102           V_VCM 0.000000 1.584000 0.000000 0.000000 0.000000
103 ;
104 param PPMOut:      CL2     EDC     HCL     VCM     ETHYL     :=
105           V_CHLO 0.000000 1.000000 0.000000 0.000000 0.000000
106           V_OXY 0.000000 1.000000 0.000000 0.000000 0.000000
107           V_VCM 0.000000 0.000000 0.584000 1.000000 0.000000
108 ;
109 param PPMCapa:      EDC_CHLO     EDC_OXY     VCM     :=
110           V_CHLO 92.730000 0.000000 0.000000
111           V_OXY 0.000000 75.090000 0.000000
112           V_VCM 0.000000 0.000000 95.570000
113 ;
114 # Les demandes
115 param Demande:      1     2     3     4     5     6     7     8     ...
116           CL2 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000  ...
117           EDC 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000  ...
118           HCL 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000  ...
119           VCM 234.743000 895.171000 896.708000 896.708000 896.708000 ...
120           ETHYL 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 ...
121 ;
122 end;
```

Données modélisant la production de soude

Les produits mentionnés dans le modèle sont :
- La saumure épurée : **SE**
- La vapeur d'eau : **VE**
- La soude légère : **SL**
- La soude dense : **SD**
- Le bicarbonate : **BIR**
- Le gaz carbonique : **CO2**

Le stockage de la saumure épurée n'étant pas important, il ne sera pas intégré dans la modélisation (la saumure épurée est intégrée uniquement parce que sa production consomme de la soude légère). La saumure est vendue à l'unité qui fait l'électrolyse et à celle de la saline. La vapeur et le CO2 ne se stockent évidemment pas.
Les paramètres Proc SNP, Proc SNP Penality, Min Proc, Min Proc Pen, Proc Batch Size et Proc Batch Pen n'interviennent pas dans le cadre d'une soudière.

Caractéristique	Valeur	Valeur	Valeur	Valeur	Valeur	Valeur
Name	SE	VE	SL	SD	BIR	CO2
StockMaxi	0	0	4300	4400	2000	0
StorageCost	0	0	0,022	0,022	0,033	0
SafetyStockLevel	0	0	4000	4000	1000	0
SafetyStockPEN	0	0	10	10	10	0
Non Del Pen	0	5000	7000	8000	10000	15000
StockInitial	0	0	4000	4000	1000	0
ProcSNPPen	NA	NA	NA	NA	NA	NA
ProcBatchSize	NA	NA	NA	NA	NA	NA
ProcBatchPen	NA	NA	NA	NA	NA	NA
MinProc	NA	NA	NA	NA	NA	NA
MinProcPen	NA	NA	NA	NA	NA	NA

TAB. 11.1 – Caractéristiques des produits

Les ressources suivantes sont utilisées :

- L'ensemble des installations servant à l'épuration de la saumure (**R_S2_SE**) ;
- Une chaudière au charbon (**R_VE_CHARB**) qui produit de la vapeur ;
- Une chaudière à gaz (**R_VE_GAZ**) qui produit également de la vapeur mais de manière plus onéreuse ;
- L'ensemble des machines qui servent à produire la soude légère (co-

lonnes de carbonation, four à chaux, filtreurs et sécheurs) (**R_SL**). Dans cette modélisation, nous ne distinguerons pas les différentes machines car cela n'apporte aucun complément d'information ;
- Un sécheur (**R_SD**) qui produit la soude dense à partir de la soude légère ;
- L'ensemble des machines servant à la production du bicarbonate (colonnes de carbonation, essoreuses, sécheurs) (**R_BIR**) ;

Caractéristique	Valeur	Valeur	Valeur	Valeur	Valeur	Valeur
Name	R_S2_SE	R_VE_CHARB	R_VE_GAZ	R_SL	R_SD	R_BIR
MaxResCap	24	24	24	24	24	24
MaxVar	20	20	20	20	20	20
ResLevIn	24	24	0	24	24	24
MinResCap	0	0	0	12	0	6
MinResPen	0	0	0	8000	0	8000

TAB. 11.2 – Caractéristiques des ressources

Dans le cadre de la soudière, nous avons toujours un seul PPM par ressource. Les coûts de production du bicarbonate sont faibles. Les coûts les plus importants sont les coûts énergétiques nécessaires à la production de vapeur d'eau. Quant aux coûts de production de la soude légère et de la soude dense, ils sont négligeables.

Caractéristique	Valeur	Valeur	Valeur	Valeur	Valeur	Valeur
Name	P_S2_SE	P_VE_CHARB	P_VE_GAZ	P_SL	P_SD	P_BIR
PPMCost	0	31380	34810	0	0	691

TAB. 11.3 – Caractéristiques des PPMs

Nous avons une capacité de production de 1500 T/J pour la SL, de 1000 T/J pour la SD et de 150 T/J pour le bicarbonate. La production de vapeur s'élève à 10.000 T/J par chaudière.

PPM	Produit	PPMIn	PPMOut
P_S2_SE	SL	5	0
P_S2_SE	SE	0	1,000
P_VE_CHARB	VE	0	2,500
P_VE_GAZ	VE	0	2,500
P_SL	SE	4,900	0
P_SL	VE	2,500	0
P_SL	SL	0	1,000
P_SL	CO2	0	1,30
P_SD	SL	1,000	0
P_SD	SD	0	1,000
P_BIR	SL	0,65	0
P_BIR	CO2	0,60	0
P_BIR	BIR	0	1,000

TAB. 11.4 – Modes opératoires

PPM	Ressource	Capacité(Secondes)
P_S2_SE	R_S2_SE	86,4
P_VE_CHARB	R_VE_CHARB	21,6
P_VE_GAZ	R_VE_GAZ	21,6
P_SL	R_SL	57,6
P_SD	R_SD	86,4
P_BIR	R_BIR	576

TAB. 11.5 – Capacités des ressources

Programme C du module

Ci-joint le code du module. Le programme est écrit en langage C.

```c
/*****************************************************************/
/*   Marc-Antoine Leroux                                         */
/* Lit un fichier XML et écrit les fichiers pour mon optimiseur */
/*****************************************************************/

    #include <stdio.h>
    #include <stdlib.h>
    #include <stdarg.h>
    #include <string.h>
    #include <expat.h>
    #include <xmlapi.h>
    #include "glpk.h"

    #define BUFFSIZE      8000


    char Buff[BUFFSIZE];
    int Depth;
    int b_minproc;
    int b_minprod;
    int b_batchsize;

/*-------------------------------------------------------------------*/
/*          Définitions de données                                   */
/*-------------------------------------------------------------------*/
// faite dans xmlapi.h

/*-------------------------------------------------------------------*/
/*          Variables globales                                       */
/*-------------------------------------------------------------------*/
// La liste des produits des ressources et des PPMS et des jours
// de l'horizon de production
PENSEMBLE produits;
PENSEMBLE ressources;
PENSEMBLE ppms;
PENSEMBLE jours;

// On stocke les données
// Relatives aux produits
//--> Product Master DATA
DATA *ssl = NULL;         // SafetyStocklevel
DATA *stockmax = NULL;    // Stock Maximum
DATA *storcost = NULL;    // Couts par unité de stockage
DATA *ssc = NULL;         // Safety Stock Cost
DATA *nondelpen = NULL;   // Non delivery penality
DATA *proccost = NULL;    // Procurement Cost
```

```
47 DATA *procsnppen = NULL; // SNP Penality
48 DATA *minprocpen = NULL; // Minimum procurement penality
49 DATA *procbatchsize = NULL; // Size of the batch
50 DATA *procbatchpen = NULL; // Penality in case of batch of wrong size
51
52 // --> Product Transactionnal DATA
53 DATA *stockinit = NULL; // Stock initial
54 DATA *procsnp = NULL;    // Procurement calculated by SNP
55 DATA *minproc = NULL;    // Procurment minimum
56 DOUBLEDATA *demin = NULL; // Demande initiale
57
58 // Relatives aux ressources
59 //--> Ressource Master DATA
60 DATA *maxrescap = NULL; // Capacité maximale
61 DATA *maxvar = NULL;     // Variation de prod maximale
62 DATA *minrescap = NULL; // Capacité minimale
63 DATA *minrespen = NULL; // Coûts en cas de franchissement de la cap. min.
64
65 // --> Ressource Transactionnal DATA
66 DATA *reslevin = NULL;   // Niveau inital de production
67 DOUBLEDATA *resinc = NULL; // Incapacités de prod sur la ressource
68
69 // Relatives aux PPMs
70 //--> PPMs Master DATA
71 DATA *cost = NULL;        // PPM Cost
72 DOUBLEDATA * ppmin = NULL; // PPM In
73 DOUBLEDATA * ppmout = NULL; // PPM Out
74 DOUBLEDATA * ppmcapa = NULL; // PPM Capa
75
76 // Les résultats
77 DOUBLEDATA *stocklevel = NULL;
78 DOUBLEDATA *satdem = NULL;
79 DOUBLEDATA *unsatdem = NULL;
80 DOUBLEDATA *proclevel = NULL;
81 DOUBLEDATA *prodlevel = NULL;
82 DOUBLEDATA *reslevel = NULL;
83
84 // Pour la justification de la fonction objectif
85 DOUBLEDATA *ppmlevel = NULL;
86 DOUBLEDATA *beta = NULL;
87 DOUBLEDATA *minproclevel = NULL;
88 DOUBLEDATA *batchpenality = NULL;
89 DOUBLEDATA *minreslevel = NULL;
90
91 /*-------------------------------------------------------------------*/
92 /*          Fonctions utiles                                         */
93 /*-------------------------------------------------------------------*/
94
95 XMLAPI char * xml_strdup (const char * str)
```

```
96  {
97     char * ret;
98     ret = malloc (strlen (str) + 1);
99     strcpy (ret, str);
100    return (ret);
101 }
102
103 XMLAPI void xml_strfree (char * str)
104 {
105    free (str);
106 }
107
108 // XML_CREATE : Créé un element vide
109 XMLAPI XML * xml_create (const char * name)
110 {
111    XML * ret;
112
113    ret = (XML *) malloc (sizeof (struct _element));
114    memset (ret, 0, sizeof (struct _element));
115    ret->name = xml_strdup (name);
116    return (ret);
117 } // fin xml_create
118
119
120 // XML_SET : Ajoute un attribut
121 XMLAPI void xml_set (XML * xml, const char * name, const char * value)
122 {
123    ATTR * attr;
124    char * holder;
125
126    if (!xml) return;
127    if (!value) value = "";
128
129    attr = xml->attrs;
130    while (attr) {
131       if (!strcmp (attr->name, name)) break;
132       attr = attr->next;
133    }
134
135    if (attr) {
136       holder = xml_strdup (value);
137       free ((void *) (attr->value));
138       attr->value = holder;
139       attr->valsize = strlen (value) + 1;
140       return;
141    }
142
143    if (xml->attrs == NULL) {
144       attr = (ATTR *) malloc (sizeof (struct _attr));
```

```
145         xml->attrs = attr;
146      } else {
147         attr = xml->attrs;
148         while (attr->next) attr = attr->next;
149         attr->next = (ATTR *) malloc (sizeof (struct _attr));
150         attr = attr->next;
151      }
152
153      attr->next = NULL;
154      attr->name = xml_strdup (name);
155      if (value) {
156         attr->value = xml_strdup (value);
157         attr->valsize = strlen (value) + 1;
158      } else {
159         attr->value = xml_strdup ("");
160         attr->valsize = 1;
161      }
162   } // Fin xml_set
163
164
165   // XML_APPEND Insere un element
166   XMLAPI void xml_append (XML * parent, XML * child)
167   {
168      ELEMENTLIST * list;
169      ELEMENTLIST * ch;
170
171      child->parent = parent;
172
173      list = (ELEMENTLIST *) malloc (sizeof(struct _list));
174      list->element = child;
175      list->prev = parent->lastchild;
176      if (list->prev) list->prev->next = list;
177      list->next = NULL;
178
179      if (parent->children == NULL) {
180         parent->children = list;
181      }
182      parent->lastchild = list;
183   } // Fin xml_append
184
185
186   // On nettoye tout
187   XMLAPI void xml_free (XML * xml)
188   {
189      ATTR * attr;
190      ELEMENTLIST * list;
191
192      if (xml == NULL) return;
193
```

```
194     if (xml->cleanup && xml->extra) {
195        (*(xml->cleanup)) (xml->extra);
196     }
197
198     if (xml->name != NULL) free ((void *) (xml->name));
199     while (xml->attrs) {
200        attr = xml->attrs;
201        xml->attrs = xml->attrs->next;
202        if (attr->name != NULL) free ((void *) (attr->name));
203        if (attr->value != NULL) free ((void *) (attr->value));
204        xml->attrs = attr->next;
205        free ((void *) attr);
206     }
207
208     while (xml->children) {
209        list = xml->children;
210        xml->children = list->next;
211        if (list->element != NULL) xml_free (list->element);
212        free ((void *) list);
213     }
214     free ((void *) xml);
215  }
216
217
218   void init_master(PENSEMBLE * ptableau)
219   {
220     ptableau->first = NULL;
221     ptableau->last = NULL;
222   }
223
224
225   void  create_master (PENSEMBLE * dblist, char * nom )
226  {
227
228     ENSEMBLE *cherche = dblist->first;
229     int trouve = 0;
230     while(cherche)
231     {
232         if (strcmp (cherche->name, nom) == 0 ) trouve = 1;
233         cherche = cherche->next;
234     }
235
236     if (trouve == 0)
237     {
238      ENSEMBLE* nouv;
239      nouv = (ENSEMBLE *) malloc(sizeof(struct _ensemble));
240
241      nouv->name = malloc( strlen(nom)* sizeof(char) );
242      strcpy(nouv->name,nom);
```

```
243     nouv->next = NULL;
244     nouv->prev = dblist->last;
245
246     if(dblist->last) dblist->last->next = nouv;
247     else dblist->first = nouv;
248
249     dblist->last = nouv;
250    }
251
252 } // Fin de create_master
253
254
255  void create_table (double j, DATA ** tableau, XML * elt, char * type )
256 {
257    XML * gpp;
258    char * noeud;
259    DATA * tab;
260
261 // J'alloue la mémoire à mon tableau
262    tab = (DATA *) malloc(sizeof(struct _data));
263
264 // La valeur de la caractéristique
265    tab->next = *tableau;
266    tab->value = j;
267
268 // Je cherche le nom de la caractéristique : Attribut du noeud grand-père
269    gpp = elt->parent;
270    noeud = gpp->name;
271    if (strcmp (noeud, type) == 0 )
272       {
273 //       tab->name = gpp->attrs->value;
274         tab->name = malloc( strlen(gpp->attrs->value)* sizeof(char) );
275         strcpy(tab->name,gpp->attrs->value);
276       }
277
278    * tableau = tab;
279 } // Fin de create_table
280
281
282  void create_double_table (double j, DOUBLEDATA ** dtableau, XML * elt,
283  char * type )
284 {
285    XML * gpp;
286    char * noeud;
287    DOUBLEDATA * dtab;
288
289 // J'alloue la mémoire à mon tableau
290    dtab = (DOUBLEDATA *) malloc(sizeof(struct _doubledata));
291
```

```
292 // La valeur de la caractéristique
293    dtab->next = *dtableau;
294    dtab->value = j;
295
296 // Je cherche le nom de la ligne : Attribut du noeud grand-père
297    gpp = elt->parent;
298    noeud = gpp->name;
299    if (strcmp (noeud, type) == 0 )
300       {
301 //       dtab->ligne = gpp->attrs->value;
302        dtab->ligne = malloc( strlen(gpp->attrs->value)* sizeof(char) );
303        strcpy(dtab->ligne,gpp->attrs->value);
304       }
305
306 // Je cherche le nom de la colonne : Attribut du noeud père
307 //   dtab->colonne = elt->attrs->value;
308    dtab->colonne = malloc( strlen(elt->attrs->value)* sizeof(char) );
309    strcpy(dtab->colonne,elt->attrs->value);
310
311    * dtableau = dtab;
312 } // Fin de create_table
313
314
315 // void print_ensemble (ENSEMBLE ** ensemble, FILE * fichier, char * param)
316  void print_ensemble (PENSEMBLE ensemble, FILE * fichier, char * param)
317 {
318 //       ENSEMBLE *tmp = *ensemble;
319        ENSEMBLE *tmp = ensemble.first;
320        fprintf(fichier,"%s",param);
321        while(tmp)
322        {
323            fprintf(fichier,"%s ", (tmp)->name);
324            tmp = (tmp)->next;
325        }
326        fprintf(fichier,";\n");
327 }
328
329  void print_simple (DATA ** tableau, FILE * fichier, char * param)
330 {
331    DATA * imprime = *tableau;
332
333    fprintf(fichier,"%s",param);
334    while (imprime)
335    {
336      fprintf(fichier,"%s %f\n",(imprime)->name,(imprime)->value);
337      imprime = (imprime)->next;
338    }
339    fprintf(fichier,";\n");
340 }
```

```
341
342
343 void print_tableau
344 (PENSEMBLE col, PENSEMBLE lig, DOUBLEDATA **val, FILE * fichier, char * param)
345 {
346
347  ENSEMBLE * tmp = col.first;
348  fprintf(fichier,"%s",param);
349  while(tmp)
350  {
351   // J'imprime le libellé des colonnes
352   fprintf(fichier,"%s    ", (tmp)->name);
353   tmp = (tmp)->next;
354  }
355  fprintf(fichier,":=\n");
356
357  ENSEMBLE *ligne = lig.first;
358  ENSEMBLE *colonne = col.first;
359  DOUBLEDATA * value = *val;
360  int trouve = 0;
361  while(ligne) // Pour toutes les lignes
362   {
363    fprintf(fichier,"         %s", ligne->name); // Le nom du ppm
364    while(colonne) // Et toutes les colonnes
365      {
366       while (value) // Je parcoure toutes les valeurs pour trouver la bonne
367        {
368         if ((strcmp(ligne->name,value->ligne) == 0) &&
369             (strcmp(colonne->name,value->colonne) == 0))
370             {
371              trouve = 1;
372              fprintf(fichier," %f ", value->value); // La valeur
373              break;
374             }
375         value = value->next;
376         }
377       value = *val;
378       // Si on a rien trouvé c'est que la valeur est 0
379       if (trouve == 0 ) fprintf(fichier," %f ", 0.0);
380       trouve = 0;
381       colonne = colonne->next;
382      }
383      colonne = col.first;
384
385      fprintf(fichier,"\n");
386      ligne = (ligne)->next;
387     }
388     fprintf(fichier,";\n");
389 }
```

```
390
391
392 void Decompose
393 (char *source, char *chA, char *chB, char *chC, char sep1, char sep2, char sep3)
394 {
395   int i;
396   int n = 1;
397   int l;
398   char car[2] ="";
399   car[1] = '\0';
400
401   strcpy (chA, "");
402   strcpy (chB,"");
403   strcpy (chC, "");
404
405   l = strlen(source);
406   for (i=0; i<l; i++) {
407     car[0] = source[i];
408
409     if ( source[i] == '\n') n = 3;
410     if ( source[i] == sep1) n = 4;
411     if ( source[i] == sep2) n = 5;
412     if (source[i] == sep3) n = 6;
413     switch (n) {
414       case 1: strcat (chA, car); break;
415       case 2: strcat (chB, car); break;
416       case 3: strcat (chC, car); break;
417       case 4: n = 2; break;
418       case 5: n = 3;
419       case 6: break;
420     } // Fin du switch
421  } // Fin du for
422 } // Fin de decompose
423
424
425 void create_result (DOUBLEDATA ** result, char * ligne, char * colonne,
426 double val)
427 {
428     DOUBLEDATA * res;
429
430 // J'alloue la mémoire à mon tableau
431    res = (DOUBLEDATA *) malloc(sizeof(struct _doubledata));
432
433    res->next = *result;
434    res->value = val;
435
436    res->ligne = malloc( strlen(ligne)* sizeof(char) );
437    strcpy(res->ligne,ligne);
438    res->colonne = malloc( strlen(colonne)* sizeof(char) );
```

```
439     strcpy(res->colonne,colonne);
440
441     * result= res;
442 } // Fin de create_result
443
444
445 void print_res_xml
446 (PENSEMBLE lig, PENSEMBLE col, DOUBLEDATA **val, FILE * fichier, char * cara)
447 {
448 char gui = '"';
449
450 // La balise de la variable
451 //fprintf(fichier,"<%s>\n",cara);
452 fprintf(fichier,"<variable> %s\n",cara);
453
454  ENSEMBLE *ligne = lig.first;
455  ENSEMBLE *colonne = col.first;
456  DOUBLEDATA * value = *val;
457  while(ligne) // Pour toutes les lignes
458   {
459    fprintf(fichier,"<ligne> %s", ligne->name);
460    while(colonne) // Et toutes les colonnes
461      {
462       while (value) // Je parcoure toutes les valeurs pour trouver la bonne
463        {
464         if ((strcmp(ligne->name,value->ligne) == 0) &&
465             (strcmp(colonne->name,value->colonne) == 0))
466             {
467              fprintf(fichier,"<colonne> %s \n <value>%f</value>\n</colonne>\n",
468                     value->colonne, value->value); // La valeur
469              break;
470             }
471         value = value->next;
472         }
473       value = *val;
474
475       colonne = colonne->next;
476      }
477      colonne = col.first;
478
479      fprintf(fichier,"</ligne>\n");
480      ligne = (ligne)->next;
481     }
482
483 // Fin de la balise de la variable
484 //fprintf(fichier,"</%s>\n",cara);
485 fprintf(fichier,"</variable>\n");
486 } // Fin de print_res_xml
487
```

```
488
489 void calcule_fo
490 (DATA **lig, DOUBLEDATA **val, FILE * fichier, char * cara)
491 {
492  DATA *master = *lig;
493  DOUBLEDATA * value = *val;
494
495  double total=0;
496  double calc=0;
497
498  while (value) // Je parcoure toutes les valeurs
499  {
500   while (master) // Le fichier des master data
501   {
502    if  (strcmp(master->name,value->ligne) == 0)
503    {
504     calc = 0;
505     calc = (master->value) * (value->value);
506     total = total + calc;
507     break;
508     }
509    master = master->next;
510    } // Fin du while (master)
511    value = value->next;
512   } // Fin de while (value)
513
514   value = *val;
515   fprintf(fichier,"Le total des %s :%f\n",cara, total);
516
517 } // Fin de calcule_fo
518
519
520 /*-------------------------------------------------------------------------*/
521 /*          Implemantation des Handlers                                    */
522 /*-------------------------------------------------------------------------*/
523
524 // Handler quand on rencontre un element de debut
525   static void XMLCALL
526   start(void *userData, const char *name, const char **attr)
527   {
528    XML ** parent;
529    XML * element;
530    element = xml_create (name);
531    while (*attr) {
532       xml_set(element, attr[0], attr[1]);
533       char * nom = xml_strdup (attr[1]);
534
535       if (strcmp (name, "produit") == 0 )
536           create_master(&produits, nom);
```

```
537       if (strcmp (name, "ressource") == 0 )
538           create_master(&ressources, nom);
539       if (strcmp (name, "ppm") == 0 )
540           create_master(&ppms, nom);
541       if (strcmp (name, "dem") == 0 )
542           create_master(&jours, nom);
543       attr += 2;
544    } //Fin du while
545
546    parent = (XML **) userData;
547    if (*parent != NULL) xml_append (*parent, element);
548    *parent = element;
549     } // Fin du Handler start
550
551 // Handler quand on rencontre un element de fin
552     static void XMLCALL
553     end(void *userData, const char *name)
554     {
555    XML ** element;
556
557    element = (XML **) userData;
558    if ((*element)->parent != NULL) *element = (*element)->parent;
559     } // Fin du Handler END
560
561
562
563 // Character Handler
564 static void XMLCALL
565 charData (void *userData, const XML_Char *s, int len) {
566
567 // Récupération des données en liste chainée
568    XML ** parent;
569    XML * elt;
570    char * noeud;
571
572    parent = (XML **) userData;
573    elt = * parent;
574
575 // Je cherche le nom du noeud
576    noeud = elt->name;
577
578 // Je Prend la valeur du champ
579    double j;
580    j = atof(s);
581
582 // Les données relatives aux produits
583 // Safety Stock Level
584    if (strcmp (noeud, "ssl") == 0 )
585        create_table (j, &ssl, elt, "produit");
```

```
586
587 // Stock maximum
588    if (strcmp (noeud, "stockmax") == 0 )
589        create_table (j, &stockmax, elt, "produit");
590
591 // Storage cost
592    if (strcmp (noeud, "storcost") == 0 )
593        create_table (j, &storcost, elt, "produit");
594
595 // Safety Stock Cost
596    if (strcmp (noeud, "ssc") == 0 )
597        create_table (j, &ssc, elt, "produit");
598
599 // Non delivery penality
600    if (strcmp (noeud, "nondelpen") == 0 )
601        create_table (j, &nondelpen, elt, "produit");
602
603 // Procurement Cost
604    if (strcmp (noeud, "proccost") == 0 )
605        create_table (j, &proccost, elt, "produit");
606
607 // SNP Penality
608    if (strcmp (noeud, "procsnppen") == 0 )
609        create_table (j, &procsnppen, elt, "produit");
610
611 // Minimum procurement penality
612    if (strcmp (noeud, "minprocpen") == 0 )
613        create_table (j, &minprocpen, elt, "produit");
614
615 // Procurement Batch size
616    if (strcmp (noeud, "procbatchsize") == 0 )
617        create_table (j, &procbatchsize, elt, "produit");
618
619 // Procurement Batch size penality
620    if (strcmp (noeud, "procbatchpen") == 0 )
621        create_table (j, &procbatchpen, elt, "produit");
622
623 // Stock Initial
624    if (strcmp (noeud, "stockinit") == 0 )
625        create_table (j, &stockinit, elt, "produit");
626
627 // Procurement calculated by SNP
628    if (strcmp (noeud, "procsnp") == 0 )
629        create_table (j, &procsnp, elt, "produit");
630
631 // Procurment minimum
632    if (strcmp (noeud, "minproc") == 0 )
633        create_table (j, &minproc, elt, "produit");
634
```

```
635 // La demande initiale
636    if (strcmp (noeud, "dem") == 0 )
637        create_double_table (j, &demin, elt, "produit");
638 // Fin des données relatives aux produits
639
640 // Les données relatives aux ressources
641 // Le niveau de ressource initial
642    if (strcmp (noeud, "reslevin") == 0 )
643        create_table (j, &reslevin, elt, "ressource");
644
645 // La capacite maximale des ressources
646    if (strcmp (noeud, "maxrescap") == 0 )
647        create_table (j, &maxrescap, elt, "ressource");
648
649 // La capacite minimale des ressources
650    if (strcmp (noeud, "minrescap") == 0 )
651        create_table (j, &minrescap, elt, "ressource");
652
653 // Coûts en cas de franchissement de la cap. min.
654    if (strcmp (noeud, "minrespen") == 0 )
655        create_table (j, &minrespen, elt, "ressource");
656
657 // Incapacités de prod sur la ressource
658    if (strcmp (noeud, "resinc") == 0 )
659        create_double_table (j, &resinc, elt, "ressource");
660
661 // La variation maximale de production pour les ressources
662    if (strcmp (noeud, "maxvar") == 0 )
663        create_table (j, &maxvar, elt, "ressource");
664 // Fin des données relatives aux ressources
665
666 // Les données relatives aux PPMS
667 // Le cout d'une PPM
668    if (strcmp (noeud, "cost") == 0 )
669        create_table (j, &cost, elt, "ppm");
670
671 // Les éléments consommés dans une ppm
672    if (strcmp (noeud, "ppmin") == 0 )
673        create_double_table (j, &ppmin, elt, "ppm");
674
675 // Les éléments produits dans une ppm
676    if (strcmp (noeud, "ppmout") == 0 )
677        create_double_table (j, &ppmout, elt, "ppm");
678
679 // Les capacités des ppms sur chaque ressource
680    if (strcmp (noeud, "ppmcapa") == 0 )
681        create_double_table (j, &ppmcapa, elt, "ppm");
682
683 // Fin des données relatives aux PPMS
```

```
684
685 // Les données relatives au fichier de paramétrage
686    if (strcmp (noeud, "b_minproc") == 0 )
687       b_minproc = j;
688
689    if (strcmp (noeud, "b_minprod") == 0 )
690       b_minprod = j;
691
692    if (strcmp (noeud, "b_batchsize") == 0 )
693       b_batchsize = j;
694
695 } // Fin de Character Handler
696
697
698 /*--------------------------------------------------------------------*/
699 /*          La Fonction MAIN                                          */
700 /*--------------------------------------------------------------------*/
701     int
702     main(int argc, char *argv[])
703     {
704      XML * ret;
705      int len;
706      int done;
707      FILE *md;
708      FILE *td;
709      FILE *fres;
710      char szBuffer;
711      FILE *data;
712      FILE *mod;
713      FILE *fjus;
714
715      init_master(&produits);
716      init_master(&ressources);
717      init_master(&ppms);
718      init_master(&jours);
719
720 /************************************************************************/
721 /* Parsing des données Maîtres                                          */
722 /************************************************************************/
723 // On créé le Parser
724       ret = NULL;
725       XML_Parser masterdata = XML_ParserCreate(NULL);
726
727       XML_SetUserData(masterdata, (void *) &ret);
728       if (! masterdata) {
729         fprintf(stderr, "Couldn't allocate memory for parser\n");
730         exit(-1);
731       }
732
```

```
733 // On lance les Handlers
734     XML_SetElementHandler(masterdata, start, end);
735     XML_SetCharacterDataHandler(masterdata, charData);
736 
737     if((md = fopen("VC_MASTER.xml", "r")) == NULL)
738     {
739      printf("Ouverture KO VC_MASTER\n");
740      exit(-1);
741     }
742 
743       while (!feof(md))
744       {
745        len = fread(Buff, 1, BUFFSIZE, md); //on lit les caractere dans le buffer
746       }
747     done = 1;
748 // On donne le fichier XML au parser
749      if (XML_Parse(masterdata, Buff, len, done) == XML_STATUS_ERROR)
750      {
751       fprintf(stderr, "Parse error Master Data at line %d:\n%s\n",
752               XML_GetCurrentLineNumber(masterdata),
753               XML_ErrorString(XML_GetErrorCode(masterdata)));
754       xml_free (ret);
755       XML_ParserFree(masterdata);
756       exit(-1);
757         }
758 
759        fclose(md);
760        printf("Parsing fini Donnees maitres\n");
761 // On libère le parser
762       XML_ParserFree(masterdata);
763 
764 
765 /*****************************************************************************/
766 /* Parsing des données transactionnelles                                     */
767 /*****************************************************************************/
768 // On créé le Parser
769       ret = NULL;
770       XML_Parser transacdata = XML_ParserCreate(NULL);
771 
772       XML_SetUserData(transacdata, (void *) &ret);
773       if (! transacdata) {
774         fprintf(stderr, "Couldn't allocate memory for parser\n");
775         exit(-1);
776       }
777 
778 // On lance les Handlers
779     XML_SetElementHandler(transacdata, start, end);
780     XML_SetCharacterDataHandler(transacdata, charData);
781 
```

```
782     if((td = fopen("VC_TRANSAC.xml", "r")) == NULL)
783     {
784      printf("Ouverture KO VC_TRANSAC\n");
785      exit(-1);
786     }
787 
788       while (!feof(td))
789       {
790        len = fread(Buff, 1, BUFFSIZE, td); //on lit les caractere dans le buffer
791       }
792     done = 1;
793 // On donne le fichier XML au parser
794      if (XML_Parse(transacdata, Buff, len, done) == XML_STATUS_ERROR)
795      {
796       fprintf(stderr, "Parse error Transac Data at line %d:\n%s\n",
797               XML_GetCurrentLineNumber(transacdata),
798               XML_ErrorString(XML_GetErrorCode(transacdata)));
799       xml_free (ret);
800       XML_ParserFree(transacdata);
801       exit(-1);
802         }
803 
804        fclose(td);
805        printf("Parsing fini Donnees transactionnelles\n");
806 // On libère le parser
807       XML_ParserFree(transacdata);
808 
809 
810 /*************************************************************************/
811 /* Parsing du fichier de paramétrisation                                 */
812 /*************************************************************************/
813 // On créé le Parser
814       ret = NULL;
815       XML_Parser paramdata = XML_ParserCreate(NULL);
816 
817       XML_SetUserData(paramdata, (void *) &ret);
818       if (! paramdata) {
819         fprintf(stderr, "Couldn't allocate memory for parser\n");
820         exit(-1);
821       }
822 
823 // On lance les Handlers
824     XML_SetElementHandler(paramdata, start, end);
825     XML_SetCharacterDataHandler(paramdata, charData);
826 
827     if((td = fopen("VC_PARAM.xml", "r")) == NULL)
828     {
829      printf("Ouverture KO VC_PARAM\n");
830      exit(-1);
```

```
831     }
832
833       while (!feof(td))
834       {
835        len = fread(Buff, 1, BUFFSIZE, td); //on lit les caractere dans le buffer
836       }
837     done = 1;
838 // On donne le fichier XML au parser
839      if (XML_Parse(paramdata, Buff, len, done) == XML_STATUS_ERROR)
840      {
841       fprintf(stderr, "Parse error Param Data at line %d:\n%s\n",
842              XML_GetCurrentLineNumber(paramdata),
843              XML_ErrorString(XML_GetErrorCode(paramdata)));
844       xml_free (ret);
845       XML_ParserFree(paramdata);
846       exit(-1);
847         }
848
849        fclose(td);
850        printf("Parsing fini Donnees de parametrage\n");
851 // On libère le parser
852       XML_ParserFree(paramdata);
853
854
855 /*************************************************************************/
856 /* Création du fichier modèle pour GLPK                                  */
857 /*************************************************************************/
858       if ((mod = fopen("model.mod","w")) == NULL)
859       {
860           printf("Ouverture KO model.mod\n");
861           exit(-1);
862       }
863        fprintf(mod,"/* Le fichier modele */\n");
864        fprintf(mod,"/* Cfr memoire Marc-Antoine Leroux */\n");
865        fprintf(mod,"\n");
866        fprintf(mod,"/*****************************************************/\n");
867        fprintf(mod,"/* Déclaration des données */\n");
868        fprintf(mod,"/*****************************************************/\n");
869        fprintf(mod,"set N; # Ensemble des produits\n");
870        fprintf(mod,"set P; # Ensemble des ressources\n");
871        fprintf(mod,"set O; # Ensemble des PPMs\n");
872        fprintf(mod,"\n");
873        fprintf(mod,"param m integer; # Nombre de jours dans l'horizon de
874               planification \n");
875        fprintf(mod,"param M := 9999; # Variable mathématique (grande)\n");
876        fprintf(mod,"param G := 9999; # Variable mathématique (grande)\n");
877        fprintf(mod,"\n");
878        fprintf(mod,"/* Les données : */\n");
879        fprintf(mod,"# Les données relatives au produit :\n");
```

```
880         fprintf(mod,"/* Niveau du Safety Stock (en Tonne)*/\n");
881         fprintf(mod,"param SafetyStockLevel{a in N};\n");
882         fprintf(mod,"\n");
883         fprintf(mod,"/* Niveau du Stock Maximum (en Tonne)*/\n");
884         fprintf(mod,"param StockMaxi{a in N};\n");
885         fprintf(mod,"\n");
886         fprintf(mod,"/* Couts pas unité de stockage */\n");
887         fprintf(mod,"param StorageCost{a in N};\n");
888         fprintf(mod,"\n");
889         fprintf(mod,"/* Couts en cas de franchissement du stock
890                de sécurité */\n");
891         fprintf(mod,"param SafetyStockCost{a in N};\n");
892         fprintf(mod,"\n");
893         fprintf(mod,"/* Couts en cas de demande non satisfaite */\n");
894         fprintf(mod,"param NonDelPen{a in N};\n");
895         fprintf(mod,"\n");
896         fprintf(mod,"/* Couts d'achat */\n");
897         fprintf(mod,"param ProcCost{a in N};\n");
898         fprintf(mod,"\n");
899         fprintf(mod,"/* Le stock initial */\n");
900         fprintf(mod,"param StockInitial{a in N};\n");
901         fprintf(mod,"\n");
902         fprintf(mod,"/* La quantitée d'achat prévue par SNP */\n");
903         fprintf(mod,"param ProcSNP{a in N};\n");
904         fprintf(mod,"\n");
905         fprintf(mod,"param ProcSNPPen{a in N};\n");
906         fprintf(mod,"\n");
907         if (b_minproc == 1)
908         {
909          fprintf(mod,"/* La quantitée minimum a acheter */\n");
910          fprintf(mod,"param MinProc{a in N};\n");
911          fprintf(mod,"\n");
912          fprintf(mod,"param MinProcPen{a in N};\n");
913          fprintf(mod,"\n");
914         }
915         if (b_batchsize == 1)
916         {
917          fprintf(mod,"/* La taille du lot à acheter */\n");
918          fprintf(mod,"param ProcBatchSize{a in N};\n");
919          fprintf(mod,"\n");
920          fprintf(mod,"param ProcBatchPen{a in N};\n");
921          fprintf(mod,"\n");
922          }
923         fprintf(mod,"\n");
924         fprintf(mod,"# Les données relatives aux ressources\n");
925         fprintf(mod,"/* Niveau initial de production sur les ressources */\n");
926         fprintf(mod,"param ResLevIn{r in P};\n");
927         fprintf(mod,"\n");
928         fprintf(mod,"/* Capacités maximales des ressources (en heure) */\n");
```

```
929         fprintf(mod,"param MaxResCap{r in P};\n");
930         fprintf(mod,"\n");
931         fprintf(mod,"/* Variation Maximale pour une ressource (en %) */\n");
932         fprintf(mod,"param MaxVar{r in P};\n");
933         fprintf(mod,"\n");
934         fprintf(mod,"/* Les périodes d'arrêts des ressources (en heure) */\n");
935         fprintf(mod,"param ResInc{r in P, j in 1..m};\n");
936         fprintf(mod,"\n");
937         if (b_minprod == 1)
938         {
939          fprintf(mod,"/* le minimum de production */\n");
940          fprintf(mod,"param MinResCap{r in P};\n");
941          fprintf(mod,"\n");
942          fprintf(mod,"param MinResPen{r in P};\n");
943          fprintf(mod,"\n");
944          }
945         fprintf(mod,"\n");
946         fprintf(mod,"# Les données relatives aux PPMs\n");
947         fprintf(mod,"/* Cout d'utilisation du PPM */\n");
948         fprintf(mod,"param PPMCost{b in O};\n");
949         fprintf(mod,"\n");
950         fprintf(mod,"/* La quantitée de l'article n Consommé par la ppm o */\n");
951         fprintf(mod,"param PPMIn{b in O, a in N};\n");
952         fprintf(mod,"\n");
953         fprintf(mod,"/* La quantitée de l'article n produite par la ppm o */\n");
954         fprintf(mod,"param PPMOut{b in O, a in N};\n");
955         fprintf(mod,"\n");
956         fprintf(mod,"/* Capacité d'un PPM (secondes) sur la ressource r */\n");
957         fprintf(mod,"param PPMCapa{b in O, r in P};\n");
958         fprintf(mod,"\n");
959         fprintf(mod,"\n");
960         fprintf(mod,"# Les demandes\n");
961         fprintf(mod,"/* La demande du produit a en période j */\n");
962         fprintf(mod,"param Demande{a in N,j in 1..m};\n");
963         fprintf(mod,"\n");
964         fprintf(mod,"\n");
965         fprintf(mod,"\n");
966         fprintf(mod,"/*****************************************************/\n");
967         fprintf(mod,"/* Déclaration des variables */\n");
968         fprintf(mod,"/*****************************************************/\n");
969         fprintf(mod,"\n");
970         fprintf(mod,"/* Quantitée stockée du produit a au jour j */\n");
971         fprintf(mod,"var stocklevel{a in N,j in 1..m}, >= 0;\n");
972         fprintf(mod,"\n");
973         fprintf(mod,"/* Demande satisfaite du produit a au jour j */\n");
974         fprintf(mod,"var satdem{a in N,j in 1..m}, >= 0;\n");
975         fprintf(mod,"\n");
976         fprintf(mod,"/* Demande non-satisfaite du produit a au jour j */\n");
977         fprintf(mod,"var unsatdem{a in N,j in 1..m}, >= 0;\n");
```

```
978         fprintf(mod,"\n");
979         fprintf(mod,"/* Demande dependante sur le produit a au jour j */\n");
980         fprintf(mod,"var depdem{a in N,j in 1..m}, >= 0;\n");
981         fprintf(mod,"\n");
982         fprintf(mod,"/* Quantitée achetée du produit a au jour j */\n");
983         fprintf(mod,"var proclevel{a in N,j in 1..m}, >= 0;\n");
984         fprintf(mod,"\n");
985         fprintf(mod,"/* Taux d'utilisation du PPM p au jour j */\n");
986         fprintf(mod,"var ppmlevel{b in 0,j in 1..m}, >= 0;\n");
987         fprintf(mod,"\n");
988         fprintf(mod,"var prodlevel{a in N,j in 1..m}, >= 0;\n");
989         fprintf(mod,"\n");
990         fprintf(mod,"var reslevel{r in P,j in 1..m}, >= 0;\n");
991         fprintf(mod,"\n");
992         fprintf(mod,"var alpha{a in N,j in 1..m}, >= 0;\n");
993         fprintf(mod,"\n");
994         fprintf(mod,"var beta{a in N,j in 1..m}, >= 0;\n");
995         fprintf(mod,"\n");
996         fprintf(mod,"var gamma{a in N}, >= 0;\n");
997         fprintf(mod,"\n");
998         fprintf(mod,"var delta{a in N}, >= 0;\n");
999         fprintf(mod,"\n");
1000        fprintf(mod,"/* Différence avec la quantitée minimum à acheter */\n");
1001        fprintf(mod,"var minproclevel{a in N,j in 1..m}, >= 0;\n");
1002        fprintf(mod,"\n");
1003        fprintf(mod,"/* Variable binaire pour l'achat minimum */\n");
1004        fprintf(mod,"var x{a in N,j in 1..m}, binary;\n");
1005        fprintf(mod,"\n");
1006        fprintf(mod,"/* Quantitée manquate pour arriver à un lot complet */\n");
1007        fprintf(mod,"var batchpenality{a in N,j in 1..m}, >= 0;\n");
1008        fprintf(mod,"\n");
1009        fprintf(mod,"/* Le nombre de lots à acheter le jour j */\n");
1010        fprintf(mod,"var numberofbatches{a in N,j in 1..m}, integer;\n");
1011        fprintf(mod,"\n");
1012        fprintf(mod,"var minreslevel{r in P,j in 1..m}, >= 0;\n");
1013        fprintf(mod,"\n");
1014        fprintf(mod,"/* Variable binaire pour la production minimum */\n");
1015        fprintf(mod,"var y{r in P,j in 1..m}, binary;\n");
1016        fprintf(mod,"\n");
1017        fprintf(mod,"\n");
1018        fprintf(mod,"/*****************************************************/\n");
1019        fprintf(mod,"/* La fonction objectif */\n");
1020        fprintf(mod,"/*****************************************************/\n");
1021        fprintf(mod,"/* Couts de production*/\n");
1022        fprintf(mod,"minimize coutTotal: sum{b in 0,j in 1..m} ppmlevel[b,j]
1023                  * PPMCost[b]\n");
1024        fprintf(mod,"\n");
1025        fprintf(mod,"/* Couts de stockage */\n");
1026        fprintf(mod,"          + sum{a in N,j in 1..m} stocklevel[a,j] *
```

```
1027            StorageCost[a]\n");
1028        fprintf(mod,"\n");
1029        fprintf(mod,"/* Couts en cas de franchissement du Safety Stock */\n");
1030        fprintf(mod,"          + sum{a in N,j in 1..m} beta[a,j]
1031            * SafetyStockCost[a]\n");
1032        fprintf(mod,"\n");
1033        fprintf(mod,"/* Couts en cas de demande non satisfaite */\n");
1034        fprintf(mod,"          + sum{a in N,j in 1..m} unsatdem[a,j] * NonDelPen[a]\n");
1035        fprintf(mod,"\n");
1036        fprintf(mod,"/* Couts d'approvisionnement */\n");
1037        fprintf(mod,"          + sum{a in N,j in 1..m} proclevel[a,j]
1038            * ProcCost[a]\n");
1039        fprintf(mod,"\n");
1040
1041        fprintf(mod,"          + sum{a in N} delta[a] * ProcSNPPen[a]\n");
1042        fprintf(mod,"\n");
1043        if (b_minproc == 1)
1044        {
1045         fprintf(mod,"          + sum{a in N,j in 1..m} minproclevel[a,j]
1046         * MinProcPen[a]\n");
1047         fprintf(mod,"\n");
1048         }
1049        if (b_batchsize == 1)
1050        {
1051         fprintf(mod,"          + sum{a in N,j in 1..m} batchpenality[a,j]
1052         * ProcBatchPen[a]\n");
1053         fprintf(mod,"\n");
1054         }
1055       if (b_minprod == 1)
1056        {
1057         fprintf(mod,"          + sum{r in P,j in 1..m} minreslevel[r,j]
1058         * MinResPen[r]\n");
1059         fprintf(mod,"\n");
1060         }
1061        fprintf(mod,";\n");
1062        fprintf(mod,"\n");
1063        fprintf(mod,"/*********************************************************/\n");
1064        fprintf(mod,"/* Les contraintes */\n");
1065        fprintf(mod,"/*********************************************************/\n");
1066        fprintf(mod,"\n");
1067        fprintf(mod,"# 1) Contraintes d'equilibre de stock. Premiere periode\n");
1068        fprintf(mod,"s.t. EqStJ1{a in N}: StockInitial[a] + proclevel[a,1]
1069        + prodlevel[a,1] =\n");
1070        fprintf(mod,"                    stocklevel[a,1] + satdem[a,1]
1071        + depdem[a,1];\n");
1072        fprintf(mod,"\n");
1073        fprintf(mod,"# 1Bis) Autres periodes\n");
1074        fprintf(mod,"s.t. EqStJM{a in N,j in 2..m}: stocklevel[a,j-1]
1075        + proclevel[a,j] + prodlevel[a,j] =\n");
```

```
1076        fprintf(mod,"                              stocklevel[a,j]
1077        + satdem[a,j] + depdem[a,j];\n");
1078        fprintf(mod,"\n");
1079        fprintf(mod,"# 2) Respect de la contrainte de stockage maximum\n");
1080        fprintf(mod,"s.t. StockMax{a in N,j in 1..m}: stocklevel[a,j]
1081        <= StockMaxi[a];\n");
1082        fprintf(mod,"\n");
1083        fprintf(mod,"# 3) Respect de la contrainte sur le Safety Stock\n");
1084        fprintf(mod,"s.t. SafetyStock{a in N,j in 1..m}: stocklevel[a,j]
1085        - SafetyStockLevel[a] = \n");
1086        fprintf(mod,"                          alpha[a,j] - beta [a,j];\n");
1087        fprintf(mod,"\n");
1088        fprintf(mod,"# 4) Contrainte d'equilibre de la demande\n");
1089        fprintf(mod,"s.t. EqDem{a in N,j in 1..m}: Demande[a,j] =
1090        satdem[a,j] + unsatdem[a,j];\n");
1091        fprintf(mod,"\n");
1092        fprintf(mod,"# 5) Respect des achats prévus par SNP\n");
1093        fprintf(mod,"s.t. ProcSNPC{a in N}: sum{j in 1..m} proclevel[a,j]
1094        - ProcSNP[a] =\n");
1095        fprintf(mod,"                    gamma[a] - delta [a];\n");
1096        fprintf(mod,"\n");
1097        fprintf(mod,"# 6) Contrainte de nomenclature : Production\n");
1098        fprintf(mod,"s.t. NomProd{a in N,j in 1..m}: prodlevel[a,j] =\n");
1099        fprintf(mod,"      sum{b in O} ppmlevel[b,j] * PPMOut[b,a];\n");
1100        fprintf(mod,"\n");
1101        fprintf(mod,"# 7) Contrainte de nomenclature : Consommation\n");
1102        fprintf(mod,"s.t. NomCons{a in N,j in 1..m}: depdem[a,j] = \n");
1103        fprintf(mod,"     sum{b in O} ppmlevel[b,j] * PPMIn[b,a];\n");
1104        fprintf(mod,"\n");
1105        fprintf(mod,"s.t. ResDet{r in P,j in 1..m}: reslevel[r,j]
1106        = sum{b in O} ppmlevel[b,j] *\n");
1107        fprintf(mod," ( PPMCapa[b,r] / 3600 );\n");
1108        fprintf(mod,"\n");
1109        fprintf(mod,"# 9) Contrainte sur le maximum de Production\n");
1110        fprintf(mod,"s.t. MaxRes{r in P,j in 1..m}: reslevel[r,j] <=
1111        MaxResCap[r] - ResInc[r,j];\n");
1112        fprintf(mod,"\n");
1113        fprintf(mod,"s.t. VarResUPJ1{r in P}: reslevel[r,1] <=\n");
1114        fprintf(mod,"                       ResLevIn[r] + ResLevIn[r] *
1115         MaxVar[r] / 100;\n");
1116        fprintf(mod,"\n");
1117        fprintf(mod,"s.t. VarResUPJM{r in P,j in 2..m}: reslevel[r,j] <=\n");
1118        fprintf(mod,"reslevel[r,j-1] + reslevel[r,j-1] * MaxVar[r] / 100;\n");
1119        fprintf(mod,"\n");
1120        fprintf(mod,"s.t. VarResDOWNJ1{r in P}: reslevel[r,1] >=\n");
1121        fprintf(mod,"ResLevIn[r] - ResLevIn[r] * MaxVar[r] / 100;\n");
1122        fprintf(mod,"\n");
1123        fprintf(mod,"s.t. VarResDOWNJM{r in P,j in 2..m}: reslevel[r,j] >=\n");
1124        fprintf(mod,"reslevel[r,j-1] - reslevel[r,j-1] * MaxVar[r] / 100;\n");
```

```
1125        fprintf(mod,"\n");
1126        if (b_batchsize == 1)
1127        {
1128         fprintf(mod,"# 12) Taille des lots // CONTRAINTE NON OBLIGATOIRE\n");
1129         fprintf(mod,"s.t. Batchsize{a in N,j in 1..m}: proclevel[a,j] =\n");
1130         fprintf(mod,"numberofbatches[a,j] * ProcBatchSize[a]
1131         - batchpenality[a,j];\n");
1132         fprintf(mod,"\n");
1133        }
1134        if (b_minproc == 1)
1135        {
1136         fprintf(mod,"s.t. MinProczero{a in N,j in 1..m}: proclevel[a,j]
1137         <= M * x[a,j];\n");
1138         fprintf(mod,"\n");
1139         fprintf(mod,"s.t. MinProcmin{a in N,j in 1..m}: proclevel[a,j] >=\n");
1140         fprintf(mod,"MinProc[a]+ M * (x[a,j]-1) - minproclevel[a,j];\n");
1141         fprintf(mod,"\n");
1142        }
1143       if (b_minprod == 1)
1144        {
1145         fprintf(mod,"s.t. MinReszeroC{r in P,j in 1..m}: reslevel[r,j]
1146         <= G * y[r,j];\n");
1147         fprintf(mod,"\n");
1148         fprintf(mod,"s.t. MinResminC{r in P,j in 1..m}: reslevel[r,j] >=\n");
1149         fprintf(mod,"MinResCap[r]+ G * (y[r,j]-1) - minreslevel[r,j];\n");
1150        }
1151        fprintf(mod,"\n");
1152        fprintf(mod,"\n");
1153
1154  // Le fin de fichier
1155        fprintf(mod,"end;\n");
1156        fclose(mod);
1157
1158  /*****************************************************************************/
1159  /* Création du fichier des données pour le modèle GLPK                       */
1160  /*****************************************************************************/
1161       if ((data = fopen("data.mod","w")) == NULL)
1162       {
1163           printf("Ouverture KO data.mod\n");
1164           exit(-1);
1165       }
1166        fprintf(data,"/* Le fichier des données */\n");
1167        fprintf(data,"data;\n"),
1168
1169  // Les ensembles
1170        fprintf(data,"# Les Ensembles\n");
1171        print_ensemble(produits,data,"set N := ");
1172        print_ensemble(ressources,data,"set P := ");
1173        print_ensemble(ppms,data,"set O := ");
```

```
1174
1175 // Le nombre de jours dans l'horizon de planification
1176       int nbjours = 0;
1177       ENSEMBLE *tmp = jours.first;
1178       while(tmp)
1179       {
1180            nbjours +=1;
1181            tmp = (tmp)->next;
1182       }
1183       fprintf(data,"param m:=%i;\n",nbjours);
1184
1185 // Les données relatives aux produits
1186       fprintf(data,"# Les donnees relatives aux produits\n");
1187       print_simple(&ssl,data,"param SafetyStockLevel:=\n");
1188       print_simple(&stockmax,data,"param StockMaxi:=\n");
1189       print_simple(&storcost,data,"param StorageCost:=\n");
1190       print_simple(&ssc,data,"param SafetyStockCost:=\n");
1191       print_simple(&nondelpen,data,"param NonDelPen:=\n");
1192       print_simple(&proccost,data,"param ProcCost:=\n");
1193       print_simple(&stockinit,data,"param StockInitial:=\n");
1194       print_simple(&procsnp,data,"param ProcSNP:=\n");
1195       print_simple(&procsnppen,data,"param ProcSNPPen:=\n");
1196       if (b_minproc == 1 )
1197       // Uniquement si on a la contrainte d'un minimum à acheter
1198       {
1199         print_simple(&minproc,data,"param MinProc:=\n");
1200         print_simple(&minprocpen,data,"param MinProcPen:=\n");
1201       }
1202       if (b_batchsize == 1 )
1203       // Uniquement si on a la contrainte de la taille des lots
1204       {
1205         print_simple(&procbatchsize,data,"param ProcBatchSize:=\n");
1206         print_simple(&procbatchpen,data,"param ProcBatchPen:=\n");
1207       }
1208
1209 // Les données relatives aux ressources
1210       fprintf(data,"# Les donnees relatives aux ressources\n");
1211       print_simple(&reslevin,data,"param ResLevIn:=\n");
1212       print_simple(&maxrescap,data,"param MaxResCap:=\n");
1213       print_simple(&maxvar,data,"param MaxVar:=\n");
1214       print_tableau(jours, ressources, &resinc, data, "param ResInc: ");
1215       if (b_minprod == 1 )
1216       // Uniquement si on a la contrainte d'un minimum de production
1217       {
1218         print_simple(&minrescap,data,"param MinResCap:=\n");
1219         print_simple(&minrespen,data,"param MinResPen:=\n");
1220       }
1221
1222 // Les données relatives aux PPMS
```

```
1223        fprintf(data,"# Les donnees relatives aux ppms\n");
1224        print_simple(&cost,data,"param PPMCost:=\n");
1225        print_tableau(produits, ppms, &ppmin, data, "param PPMIn:  ");
1226        print_tableau(produits, ppms, &ppmout, data, "param PPMOut: ");
1227        print_tableau(ressources, ppms, &ppmcapa, data, "param PPMCapa: ");
1228
1229  // Les données relatives à la demande
1230        fprintf(data,"# Les demandes\n");
1231        print_tableau(jours, produits, &demin, data, "param Demande: ");
1232
1233  // Le fin de fichier
1234        fprintf(data,"end;\n");
1235        fclose(data);
1236
1237  /*****************************************************************************/
1238  /* Programation Linéaire : GLPK                                              */
1239  /*****************************************************************************/
1240
1241  printf("b_minproc : %d\n" , b_minproc);
1242  printf("b_minprod : %d\n" , b_minprod);
1243  printf("b_batchsize : %d\n" , b_batchsize);
1244
1245  LPX *lp;
1246  int i;
1247  int num_cols;
1248  int cpt;
1249  char* name_cols;
1250  double sol;
1251
1252  char var[133];
1253  char ligne[133];
1254  char col[133];
1255  char *cherch;
1256  char sep1 = '[';
1257  char sep2 = ',';
1258  char sep3 =']';
1259
1260
1261  //lp = lpx_read_model("mod_nodata.mod", "data.mod", NULL);
1262  lp = lpx_read_model("model.mod", "data.mod", NULL);
1263  i = lpx_simplex(lp);
1264  //i = lpx_print_prob(lp,"ennonce.txt");
1265  i = lpx_print_sol(lp,"solution.txt");
1266  if  ( b_minproc == 1 ||
1267       b_minprod == 1 ||
1268       b_batchsize == 1 )
1269  {
1270   i = lpx_integer(lp);
1271   i = lpx_print_mip(lp,"sol_entier.txt");
```

```
1272 }
1273
1274 num_cols = lpx_get_num_cols(lp);
1275
1276 for (cpt=1; cpt<=num_cols; cpt++)
1277 {
1278  name_cols = lpx_get_col_name(lp,cpt);
1279
1280
1281 if  ( b_minproc == 1 ||
1282       b_minprod == 1 ||
1283       b_batchsize == 1 )
1284 {
1285       sol = lpx_mip_col_val(lp,cpt);
1286 }
1287 else
1288 {
1289       sol = lpx_get_col_prim(lp,cpt);
1290 }
1291
1292  Decompose (name_cols, var, ligne, col, sep1, sep2, sep3);
1293
1294  if (strcmp(var,"stocklevel") == 0)
1295  create_result (&stocklevel, ligne, col, sol);
1296
1297  if (strcmp(var,"satdem") == 0)
1298  create_result (&satdem, ligne, col, sol);
1299
1300  if (strcmp(var,"unsatdem") == 0)
1301  create_result (&unsatdem, ligne, col, sol);
1302
1303  if (strcmp(var,"proclevel") == 0)
1304  create_result (&proclevel, ligne, col, sol);
1305
1306  if (strcmp(var,"prodlevel") == 0)
1307  create_result (&prodlevel, ligne, col, sol);
1308
1309  if (strcmp(var,"reslevel") == 0)
1310  create_result (&reslevel, ligne, col, sol);
1311
1312 // Pour la justification du résultat
1313  if (strcmp(var,"ppmlevel") == 0)
1314  create_result (&ppmlevel, ligne, col, sol);
1315
1316  if (strcmp(var,"beta") == 0)
1317  create_result (&beta, ligne, col, sol);
1318
1319  if (strcmp(var,"minproclevel") == 0)
1320  create_result (&minproclevel, ligne, col, sol);
```

```
1321
1322  if (strcmp(var,"batchpenality") == 0)
1323  create_result (&batchpenality, ligne, col, sol);
1324
1325  if (strcmp(var,"minreslevel") == 0)
1326  create_result (&minreslevel, ligne, col, sol);
1327
1328 } // Fin du for
1329
1330 // On libere la memoire
1331 lpx_delete_prob(lp);
1332
1333
1334 // On créé le fichier XML
1335       char gui = '"';
1336
1337       if ((fres = fopen("result.xml","w")) == NULL)
1338       {
1339           printf("Ouverture KO result.xml\n");
1340           exit(-1);
1341       }
1342        fprintf(fres,"<?xml version=%c1.0%c ?>\n",gui,gui);
1343        fprintf(fres,"\n"),
1344        fprintf(fres,"<?xml-stylesheet type=%ctext/xsl%c href=%cresult.xsl%c?>\n",
1345        gui, gui, gui, gui);
1346        fprintf(fres,"<solution>\n");
1347        print_res_xml(produits, jours, &stocklevel, fres,
1348                    "Niveau de stock");
1349        print_res_xml(produits, jours, &satdem, fres,
1350                    "Demande satisfaite");
1351        print_res_xml(produits, jours, &unsatdem, fres,
1352                    "Demande non satisfaite");
1353        print_res_xml(produits, jours, &proclevel, fres,
1354                    "Quantite a acheter");
1355        print_res_xml(produits, jours, &prodlevel, fres,
1356                    "Niveau de production");
1357        print_res_xml(ressources, jours, &reslevel, fres,
1358                    "Niveau d'utilisation d'une ressource");
1359        fprintf(fres,"</solution>");
1360        fclose(fres);
1361
1362
1363 /*************************************************************************/
1364 /* Justification de la fonction objectif                                 */
1365 /*************************************************************************/
1366
1367       if ((fjus = fopen("justif.txt","w")) == NULL)
1368       {
1369           printf("Ouverture KO justif.txt\n");
```

```
1370            exit(-1);
1371        }
1372        fprintf(fjus,"Justification de la fonction objectif\n");
1373        fprintf(fjus,"\n");
1374
1375  // Cout de production
1376        calcule_fo(&cost, &ppmlevel, fjus,
1377                   "Prod. Cost");
1378  // Cout de stockage
1379        calcule_fo(&storcost, &stocklevel, fjus,
1380                   "Stock Cost");
1381
1382  // Couts en cas de franchissement du Safety Stock
1383        calcule_fo(&ssc, &beta, fjus,
1384                   "Safety Stock Pen");
1385
1386  // Couts en cas de demande non satisfaite
1387        calcule_fo(&nondelpen, &unsatdem, fjus,
1388                   "Unsat Dem Pen");
1389
1390  // Couts d'approvisionnement
1391        calcule_fo(&proccost, &proclevel, fjus,
1392                   "Proc Cost");
1393
1394  // Couts en cas de non respect des quantités données pas SNP
1395  // Ne se caclusle pas de la même facon car Delta sur une variable
1396  //DATA *procsnppen = NULL; // SNP Penality
1397
1398       if  ( b_minproc == 1 )
1399       {
1400         calcule_fo(&minprocpen, &minproclevel, fjus,
1401                    "Min Proc Pen");
1402       }
1403
1404       if  ( b_batchsize == 1 )
1405       {
1406         calcule_fo(&procbatchpen, &batchpenality, fjus,
1407                    "Batch Size Pen");
1408       }
1409
1410       if  ( b_minprod == 1 )
1411       {
1412         calcule_fo(&minrespen, &minreslevel, fjus,
1413                    "Min Ressource Level Pen");
1414       }
1415
1416       return 0;
1417     } // Fin main
```